2019

Design and Analysis of Efficient Parallel Bayesian Model Comparison Algorithms

Robert Wesley Henderson
University of Mississippi

DESIGN AND ANALYSIS OF EFFICIENT PARALLEL BAYESIAN MODEL COMPARISON ALGORITHMS

A Dissertation
presented in partial fulfillment of requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering
The University of Mississippi

by

Robert Wesley Henderson

May 2019

ABSTRACT

A common task in science and engineering is evaluating how well a mathematical model describes a set of observations. Bayesian model comparison provides a rational and consistent method for applying logic and probability to the problem of evaluating models. Model comparison requires numerical techniques that are usually very time consuming to run. This dissertation proposes extensions to several existing numerical model comparison techniques, including nested sampling and thermodynamic integration, that incorporate parallel algorithm design to achieve significant speed-ups. Serial computer performance gains have slowed in recent years, and most processing speed improvements are seen in the area of parallel architectures. This work discusses the design, theoretical analysis, and empirical analysis of these algorithms, focusing on the performance of these algorithms with respect to accuracy and run time. Many disciplines in science and engineering make use of existing model comparison techniques. This work aims to save investigators in these disciplines time, and potentially attract those who may have been put off by time complexity concerns, by developing a general approach to model comparison that takes full advantage of modern parallel computing platforms.

DEDICATION

To the memory of Alice Trisler

Grandmother and steadfast supporter

LIST OF ABBREVIATIONS

NS-CC Combined-chain nested sampling

NS-MR Multiple-replacement nested sampling

TI Thermodynamic integration

BSS Binary slice sampling

TI-Stan Thermodynamic integration with Stan

TI-BSS Thermodynamic integration with binary slice sampling

TI-BSS-H Thermodynamic integration with binary slice sampling and the Hilbert curve

TI-BSS-Z Thermodynamic integration with binary slice sampling and the Z-order curve

HMC Hamiltonian Monte Carlo

NUTS No U Turn Sampler

MCMC Markov chain Monte Carlo

CDF cumulative distribution function

ACKNOWLEDGEMENTS

I would like to thank Prof. Ramanarayanan Viswanathan and the department of Electrical Engineering at the University of Mississippi, along with the University of Mississippi Graduate School, for funding and supporting my doctoral studies. I would also like to thank Prof. Paul Goggans, my advisor, for his guidance, support, and friendship throughout this entire process.

I would like to thank the members of my dissertation committee for their support and guidance. I would like to thank Prof. Lei Cao for co-authoring the paper on combined-chain nested sampling and helping us work through the theoretical aspects of that work. Thanks to Prof. John Daigle as well for advice on developing the theoretical framework for that work. I would also like to thank Prof. Yixin Chen for his guidance on the fundamentals of algorithm analysis, which plays an important role in this dissertation.

I would like to thank Dr. John Skilling for the initial inspiration for combined-chain nested sampling, for our conversations at MaxEnt, and for feedback on my presentations up to now. I owe thanks to Prof. Ning Xiang for sparking my interest in the area of Bayesian inference in the first place and for supporting my first contributions to the field. I also owe much to the organizers and regular attendees of the MaxEnt conference. I have drawn an immense amount of inspiration and encouragement from presentations and conversations at these meetings.

I cannot thank my wife, Mandy Henderson, enough for her immeasurable love, support, and patience throughout this process. I am so lucky you agreed to go on this journey with me. Finally, I would like to thank my family (in Ruston, Effie, and beyond), and my parents specifically, for their support and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This dissertation describes the work I have done throughout my doctorate on the design and analysis of parallel Bayesian model comparison algorithms. This work comprises the development and testing of several algorithms based on nested sampling and thermodynamic integration. These algorithms have been tested on a range of problems, both artificial and practical, to demonstrate their effectiveness. These algorithms are novel in that they incorporate parallelism in ways that the original implementations did not, thereby achieving significant speed-up.

Model comparison is one level of Bayesian inference; other levels of inference include parameter estimation and experimental design. Broadly speaking, Bayesian model comparison provides a framework for rationally and consistently ranking proposed models for a set of data using probability theory. Model rankings can then be used to make decisions informed by the data.

Model probabilities are usually impossible to compute analytically and often difficult to compute numerically. Specialized techniques have been developed to numerically evaluate the integrals necessary to obtain model probabilities. These techniques are computationally intensive, and therefore there is a strong incentive to make them as time-efficient as possible.

This dissertation collects and extends several of the projects I have worked on by refining or completing the techniques, exploring and developing new techniques, and applying rigorous algorithm analysis methods to them.

1.1 The Bayesian View of Probability

The Bayesian school of probability makes use of the same fundamental theorems and mathematics as the traditional (frequentist) school. The Bayesian view differs in its interpretation of what a probability is, and, hence, in what probabilities can be used to quantify. In the frequentist interpretation, a probability expresses the frequency with which a random event occurs, and that frequency is an inherent property of the phenomenon being observed. In the Bayesian interpretation, probability is a useful way for observers to express their state of knowledge of a phenomenon in the presence of uncertainty. In this view, probability is a property of the observer, and not the phenomenon being observed. Additionally, Bayesians view probability as an extension of deductive logic into the realm of uncertainty, and so we can discuss the probability of any logical proposition, whereas traditional random variables are limited to describing the outcomes of random events.

The following derivation of the product rule, sum rule, and Bayes' theorem follows Gregory [2010, Chapter 2]. For the purpose of this section, let a capital letter (e.g., A) represent a logical proposition, e.g.,

$$A \equiv \text{“There are three cookies in the jar.”}$$

A's value is **1** if the proposition is true, and it is **0** if the proposition is false. A, B represents a logical conjunction, i.e., A, B is true if both A and B are true individually, but it is false otherwise. $A + B$ represents a logical disjunction, i.e., $A + B$ is true if either A or B, or both, are true individually; if both are false, $A + B$ is false. $\overline{A}$ represents a logical complement, i.e., $\overline{A}$ is true only if A is false.

These operations, with the addition of some further derived operations, are sufficient for applying deductive logic, which can be applied if there is no uncertainty as to the truth of the propositions involved. In the presence of uncertainty, we need to go a little further. Cox [1946], Pólya [1954], and Jaynes [1957] lay out three desiderata that an ideal system for

performing inductive reasoning would obey. These desiderata are,

Comparability Real numbers should be used to represent degrees of plausibility.

Rationality If evidence is added in favor of a proposition, the number representing its plausibility should increase; if evidence is added in favor of a proposition's complement, the number representing its plausibility should decrease. In the limits of certainty, the value should be 1 if the proposition is true, and 0 if the proposition is false.

Consistency "If a conclusion can be reasoned out in more than one way, every possible way must lead to the same result" [Gregory, 2010, Chapter 2.5].

With the foundations of deductive logic, these desiderata, and no other starting assumptions, we can now derive the product and sum rules of probability.

Let us consider the proposition "if C is true, then A and B are true", $(A, B|C)$. There are eleven unique ways to write the relationship between that conditional proposition and pairs of related conditional propositions. As Gregory [2010, 2.5.2] shows, only two of those eleven pairs actually make sense logically,

$$(A, B|C) = F[(B|C), (A|B, C)] = F[(A|C), (B|A, C)], \tag{1.1}$$

where $F[\cdot, \cdot]$ is an arbitrary monotonic function of its arguments.

Now, let us apply (1.1) to the proposition $(A, B, C|D)$. This can be written one of two ways:

$$(A, B, C|D) = F[(B, C|D), (A|B, C, D)] = F\{F[(C|D), (B|C, D)], (A|B, C, D)\}, \tag{1.2}$$

$$= F[(C|D), (A, B|C, D)] = F\{(C|D), F[(B|C, D), (A|B, C, D)]\}. \tag{1.3}$$

Let $x \equiv (A|B, C, D)$, $y \equiv (B|C, D)$, and $z \equiv (C|D)$. Applying the consistency desideratum, we have

$$F\{x, F[y, z]\} = F\{F[x, y], z\}. \tag{1.4}$$

Gregory [2010, 2.5.2] notes that the lengthy solution to this functional equation is available in [Aczél, 1966]. The final result is

$$w\{F[x,y]\} = w\{x\}w\{y\}, \tag{1.5}$$

for any positive, continuous, and monotonic $w\{\cdot\}$.

Applying the solution (1.5) to the relation (1.4) for the proposition $(A, B|C)$ (and dropping the braces for concision),

$$w(A,B|C) = w(A|B,C)w(B|C), \tag{1.6}$$

$$= w(B|A,C)w(A|C). \tag{1.7}$$

This is commonly known as the product rule. At this point, the only restrictions on $w(\cdot)$ are that it is positive, continuous, and monotonic. The rationality desideratum allows us put some limits on the function's range in extreme cases.

First, let us consider the case where A is certainly true assuming C is also true. In this situation, logic dictates that the following relations must hold,

$$(A,B|C) = (B|C), \tag{1.8}$$

$$(A|B,C) = (A|C), \tag{1.9}$$

and the product rule yields

$$w(B|C) = w(A|C)w(B|C). \tag{1.10}$$

Equation (1.10) can only hold true if $w(A|C) = 1$. In other words, if the proposition is certain, $w(\cdot)$ must equal 1.

Now, let us consider the opposite case where A is certainly false assuming C is true.

The relations we have in this scenario are

$$(A, B|C) = (A|C), \tag{1.11}$$

$$(A|B, C) = (A|C), \tag{1.12}$$

and the product rule yields

$$w(A|C) = w(A|C)w(B|C). \tag{1.13}$$

Equation (1.13) can only be true if $w(A|C) = 0$ or $w(A|C) = \infty$. Either choice is valid, so we choose $w(A|C) = 0$ for convenience. In other words, if the proposition is certainly false, $w(\cdot)$ must equal 0.

So far, we have applied all of three of the desiderata and obtained the product rule and limits on the function $w(\cdot)$,

$$0 \leq w(x) \leq 1.$$

Further application of the rules of logic will yield the sum rule. Jaynes [2003] shows that for positive m,

$$w^m(A|B) + w^m(\overline{A}|B) = 1. \tag{1.14}$$

Equation (1.14) is the simplest form of the sum rule. Because $w(\cdot)$ is a positive, monotonically increasing function bounded by 0 and 1, the product rule also holds true for $w^m(A, B|C)$. If we define $p(x) = w^m(x)$, the product and sum rules become

$$p(A, B|C) = p(A|B, C)p(B|C) = p(B|A, C)p(A|C), \tag{1.15}$$

$$p(A|B) + p(\overline{A}|B) = 1. \tag{1.16}$$

This form of the sum rule is not very immediately useful; we would like to have an expression for $p(A + B|C)$. Table 1.1 shows the steps to find a more useful form of the sum rule side-by-side with the principles that justify each step. The resulting form of the sum

Table 1.1: Derivation of a more useful form of the sum rule

$p(A+B\|C) =$	Justification
$1 - p(\overline{A+B}\|C)$	Sum rule
$1 - p(\overline{A}, \overline{B}\|C)$	De Morgan's laws
$1 - p(\overline{A}\|C)p(\overline{B}\|\overline{A}, C)$	Product rule
$1 - p(\overline{A}\|C)[1 - p(B\|\overline{A}, C)]$	Sum rule
$1 - p(\overline{A}\|C) + p(\overline{A}\|C)p(B\|\overline{A}, C)$	Distributive law of multiplication
$p(A\|C) + p(\overline{A}, B\|C)$	Product and sum rules
$p(A\|C) + p(B\|C)p(\overline{A}\|B, C)$	Product rule
$p(A\|C) + p(B\|C)[1 - p(A\|B, C)]$	Sum rule
$p(A\|C) + p(B\|C) - p(B\|C)p(A\|B, C)$	Distributive law of multiplication
$p(A\|C) + p(B\|C) - p(A, B\|C)$	Product rule

rule from Table 1.1 is

$$p(A + B|C) = p(A|C) + p(B|C) - p(A, B|C). \tag{1.17}$$

With the product and sum rules now in hand, we can derive Bayes' Theorem. Assume that we have a set of exhaustive and mutually exclusive hypotheses H indexed by i, prior information I, and observations D. The sum rule gives

$$\sum_i p(H_i|I) = 1, \tag{1.18}$$

and the product rule gives

$$p(H_i, D|I) = p(H_i|I)p(D|H_i, I) = p(D|I)p(H_i|D, I). \tag{1.19}$$

Solving (1.19) for $p(H_i|D, I)$,

$$p(H_i|D, I) = \frac{p(H_i|I)p(D|H_i, I)}{p(D|I)}, \tag{1.20}$$

which is one of the usual forms of Bayes' Theorem. We note that because the hypotheses are exhaustive and mutually exclusive, the sum rule requires that

$$\sum_i p(H_i|D,I) = \sum_i \frac{p(H_i|I)p(D|H_i,I)}{p(D|I)} = 1, \tag{1.21}$$

hence,

$$p(D|I) = \sum_i p(H_i|I)p(D|H_i,I). \tag{1.22}$$

This yields an alternate form of Bayes' theorem,

$$p(H_i|D,I) = \frac{p(H_i|I)p(D|H_i,I)}{\sum_i p(H_i|I)p(D|H_i,I)}. \tag{1.23}$$

To conclude, using the rules of deductive logic and three common-sense desiderata, we have derived the fundamental relations of probability theory:

Product rule	$p(A,B\|C) = p(A\|B,C)p(B\|C) = p(B\|A,C)p(A\|C)$
Sum rule	$p(A+B\|C) = p(A\|C) + p(B\|C) - p(A,B\|C)$
Bayes' Theorem	$p(H_i\|D,I) = \frac{p(H_i\|I)p(D\|H_i,I)}{p(D\|I)}$

1.2 Bayesian Model Comparison

Now that we have derived Bayes' Theorem, we can turn our attention to model comparison. M_i is the proposition that model number i, $\boldsymbol{g}_i(t,\boldsymbol{\Theta}_i)$, is operative in the measured data $\boldsymbol{D}$. The model function $\boldsymbol{g}_i(t,\boldsymbol{\Theta}_i)$ is vector-valued, t is its independent variable, and $\boldsymbol{\Theta}_i$ is the vector of model parameters associated with M_i. We assume that an exhaustive and mutually exclusive set of models M_i for data $\boldsymbol{D}$ and prior information I can be enumerated, such that,

$$\sum_i p(M_i|\boldsymbol{D},I) = 1. \tag{1.24}$$

It is important to note that we do not assume that *we* can enumerate all possible models for the data, just that they exist. Writing Bayes' theorem for the parameters of M_i gives

$$\underbrace{p(\boldsymbol{\Theta}_i|M_i,\boldsymbol{D},I)}_{\text{Posterior}} = \frac{\overbrace{p(\boldsymbol{\Theta}_i|M_i,I)}^{\text{Prior}}\overbrace{p(\boldsymbol{D}|M_i,\boldsymbol{\Theta}_i,I)}^{\text{Likelihood}}}{\underbrace{p(\boldsymbol{D}|M_i,I)}_{\text{Evidence or model likelihood}}}. \tag{1.25}$$

Each probability in (1.25) is labeled with the name it is usually assigned. The posterior probability for the models can be written similarly,

$$\overbrace{p(M_i|\boldsymbol{D},I)}^{\text{Model posterior}} = \frac{\overbrace{p(M_i|I)}^{\text{Model prior}}\overbrace{p(\boldsymbol{D}|M_i,I)}^{\text{Evidence}}}{p(\boldsymbol{D}|I)}. \tag{1.26}$$

Note that the evidence, $p(\boldsymbol{D}|M_i,I)$, appears in both (1.25) and (1.26).

To find $p(\boldsymbol{D}|I)$, we would need to be able to enumerate all possible models for the data. In virtually all situations, this is not possible; however, it is also usually unnecessary. Because we are interested in comparing models, pairwise ratios of model posteriors are just as useful as the model posteriors themselves. As long as $p(\boldsymbol{D}|I)$ exists and is finite, the odds of model j versus model i are

$$O_{ji} = \frac{p(M_j|\boldsymbol{D},I)}{p(M_i|\boldsymbol{D},I)} = \underbrace{\frac{p(M_j|I)}{p(M_i|I)}}_{\text{Prior odds}} \times \underbrace{\frac{p(\boldsymbol{D}|M_j,I)}{p(\boldsymbol{D}|M_i,I)}}_{\text{Evidence ratio}}. \tag{1.27}$$

If our prior knowledge of the models leads us to assign a prior distribution to them other than a uniform distribution, we will need to use (1.27) directly to compute model odds. However, if we are able to assign a uniform prior to the model prior, the prior odds becomes 1, and the model odds simply becomes a ratio of the model evidence values,

$$O_{ji} = \frac{p(\boldsymbol{D}|M_j,I)}{p(\boldsymbol{D}|M_i,I)}. \tag{1.28}$$

Returning to (1.25), the posterior must be a proper probability distribution, i.e., it must integrate (if continuous) or sum (if discrete) to 1. Assuming the continuous case, the evidence must then be

$$p(\boldsymbol{D}|M_i, I) = \int_{\tilde{\boldsymbol{\Theta}}} p(\boldsymbol{\Theta}_i|M_i, I)p(\boldsymbol{D}|M_i, \boldsymbol{\Theta}_i, I)\,\mathrm{d}\boldsymbol{\Theta}, \tag{1.29}$$

where $\tilde{\boldsymbol{\Theta}}$ is the support of the posterior distribution. To compute model odds, we must be able to directly compute or estimate (1.29). This is usually impossible to do exactly and difficult to do numerically, especially in a time-efficient manner. Models of interest can have on the order of 10 or 100 parameters, and the associated distributions tend to have most of their mass concentrated in a relatively small portion of the parameter space. As an example, a model with fifty parameters that requires 1000 grid points per dimension to capture the detail of the distribution properly will require 10^{150} grid points in total, which would require 8×10^{150} bytes to store as double-precision values. This is clearly absurd, as a standard modern desktop computer has on the order of 10^9 bytes of memory and on the order of 10^{12} bytes of storage space. This problem is the primary motivation of this work.

1.3 Parallel Computing

This work involves designing and implementing parallel algorithms for model comparison, so a brief history and description of parallel computing is in order. Barney [2018] at Lawrence Livermore National Laboratory's Livermore Computing Center website gives an excellent tutorial on parallel computing, and this section will draw from that tutorial.

The simplest and oldest way to run programs is on a serial computer. A serial computer has one processing unit, and it can execute one instruction at a time on a single piece of data. A serial computer is predictable: it runs the instructions given in a program in order, one at a time. Assuming sufficient memory and sufficiently fast IO, making a serial computer faster is a matter of increasing the speed at which the processor can execute

instructions.

Some computing tasks have elements that do not need to be run in order. For instance, if your program does payroll for a number of employees, the payroll calculations for one employee are likely independent of the calculations for another employee. Therefore, these tasks can be completed in any order, or even at the same time, and the results will still be valid. This is a simple example of a program that could benefit from running on a parallel computer instead of a serial computer.

In contrast to a serial computer, a parallel computer can do more than one thing at a time. There are several conceptual models for parallel computing, hence "do[ing] more than one thing at once" can mean different things depending on which model is being implemented. The first widely adopted technology to implement a parallel model of computing for consumers was Intel's MMX extensions to the x86 architecture [Ch and rasekaran, 1997, Yu, 1997]. The MMX extensions enabled single-instruction, multiple-data (SIMD) parallelism, in which one instruction could be executed simultaneously on multiple data. Computational tasks that can use SIMD hardware in this way are also sometimes called "data parallel." Examples of hardware that implements an SIMD model include general purpose graphics processing units (GPGPUs) and the AVX and SSE units on Intel CPUs.

While SIMD parallelism can speed up many computing tasks, tasks that cannot be vectorized require a different model to see any benefit. The multiple-instruction, multiple-data (MIMD) model accommodates these tasks. Hardware that implements a MIMD model can execute different instructions on several data, simultaneously. Computational tasks that require MIMD hardware are sometimes called "task parallel." Examples of hardware that implements an MIMD model include multi-core processors, multi-processor systems, and networked clusters.

The programs described in this dissertation make use of SIMD and MIMD parallelism. The SIMD portions take the form of NumPy code, which automatically makes use of the vector units (i.e., SSE or AVX registers) on the CPU. The MIMD portions take the form of

invocations of Python `multiprocessing` package that facilitates process-based parallelism on a multi-core or multi-CPU system. It is worth noting here that thread-based parallelism is not possible in most Python implementations, and this is the reason that process-based parallelism is used here.

1.4 Literature Review

There are several schools of thought regarding the interpretation of probability. This work concerns itself with what is sometimes called the objective Bayesian (usually simply "Bayesian") school of interpretation. Broadly speaking, Bayesians view probability as a measure of the state of knowledge (information) regarding a given proposition. Bayesian probability is an extension of deductive logic into situations where the truth of propositions cannot be determined with certainty.

Gregory [2010, section 2.3] provides an excellent history of Bayesian inference, briefly summarized here. Bernoulli [1713], Bayes [1763], and Laplace [1774] provided the foundations for the field that would eventually be called Bayesian inference. However, Cox [1946], Pólya [1954], and Jaynes [1957] were the first to provide formal arguments for the Bayesian interpretation of probability.

Computing probabilities and moments of probability distributions requires the evaluation of integrals that are often unsuited to evaluation using analytic or standard numeric techniques. Markov chain Monte Carlo (MCMC) is a numerical integration technique that is well-suited to evaluating the integrals that appear in Bayesian computations. MCMC is fundamentally based on work in chemical physics by Metropolis et al. [1953], who developed a Monte Carlo integration method to investigate the "properties of any substance which may be considered as composed of interacting individual molecules." This Monte Carlo integration method was later generalized and expanded by Hastings [1970]. Hastings describes an approach that samples a probability distribution. The resulting samples can be used to estimate moments of the distribution.

The work of Metropolis et al. and Hastings is useful both for directly doing Bayesian parameter estimation and as a base for developing more complex and powerful parameter estimation tools, but it is not sufficient for Bayesian model comparison. Additional numerical techniques were developed to address this lack.

Thermodynamic integration, a statistical mechanics technique originally developed by Kirkwood [1935] to compute the free energy of fluids, has been adapted to aid in Bayesian model comparison. Kirkpatrick et al. [1983], Černý [1985], and Semenovskaya et al. [1985] each independently developed an optimization technique now called simulated annealing that was inspired by Kirkwood's thermodynamic integration. In a technical report, Neal [1993, Section 6.2] noted that this free energy computation technique could be adapted to compute the Bayesian evidence for model comparison. Meanwhile in mathematics, Ogata [1989] independently developed a very similar technique for numerically evaluating high-dimensional integrals, provided the integrand is continuous and differentiable. Gelman and Meng [1998] recognized the utility of the thermodynamic integration technique as described by both Ogata [1989] and Neal [1993]. Their work presented the technique, including specific implementation details, to a broader statistics audience. More recently, thermodynamic integration has been extended in various ways in the statistics literature, with examples being found in Friel and Pettitt [2008], Calderhead and Girolami [2009], and Oates et al. [2016].

All of the work on thermodynamic integration mentioned until now has used a fixed temperature schedule to estimate the the thermodynamic integral. For instance, according to Oates et al. [2016], a quintic schedule seems to produces good results for the problems considered in that paper. However, if we consider problems more complex than the relatively numerically straightforward problems presented in these papers, thermodynamic integration implementations with fixed temperature schedules tend to fail to estimate the model evidence within an acceptable amount of error. For thermodynamic integration to be successful when considering likelihoods that involve the computation of a physical parameterized

model of data, an adaptive temperature schedule is required. John Skilling's BayeSys software [Skilling, 2004a] implements thermodynamic integration with an adaptive temperature schedule in this way. Skilling gave an unpublished presentation on the topic at the MaxEnt meeting in 2002, and with his permission, Goggans and Chi [2004] wrote and published a formal write-up of the technique. The technique described by Goggans and Chi [2004] will be extended in several ways in Chapter 3.

Nested sampling was developed by Skilling [2004b, 2006] as an alternative method for computing model evidence. Sivia and Skilling [2006, Chapter 9] explain nested sampling in an accessible way. Nested sampling has been expanded in various ways in the years since its introduction. Feroz et al. [2009] introduced MultiNest, an adaptation of nested sampling made specifically to handle multi-modal likelihood functions. Diffusive nested sampling, developed by Brewer et al. [2011], allows for the sampling of a mixture of distributions instead of only the prior constrained by a single likelihood constraint (as in classic nested sampling) to replace the discarded sample. Burkoff et al. [2012], Henderson and Goggans [2014], and Henderson et al. [2017] describe two ways to parallelize the nested sampling algorithm. Finally, Handley et al. [2015] describes an algorithm that combines nested sampling, slice sampling, and clustering to efficiently sample distributions that are high-dimensional and multi-modal.

Thermodynamic integration and nested sampling are approaches that evaluate model probabilities for one model at a time. An alternative approach is to evaluate each model under consideration simultaneously. Green [1995] was the first to propose such a technique, known as reversible jump Markov chain Monte Carlo (RJMCMC). RJMCMC treats the model index as a parameter and explores the space formed by the union of the model parameter spaces. Under this scheme, the estimated model probabilities are proportional to the sample counts for each model in the final Markov chain.

If the family of models under consideration has atomic prior distributions (i.e., if the model equation is composed of the sum of some number of addend functions, and if each

addend function's parameters have identical prior distributions), a more specific method can be used to perform simultaneous model comparison. Phillips and Smith [1996] describe a so-called jump-diffusion method for evaluating these types of model. This method has several different moves available at a given MCMC step: birth, death, and within-model. A birth move adds an atomic component to the model function, a death move removes an atomic component from the model function, and a within-model move allows one or more the current atom's parameters to move about the parameter space. John Skilling's BayeSys software [Skilling, 2004a] incorporates jump diffusion sampling as part of its model comparison scheme.

1.5 Contribution and Outline

This dissertation fits into the current literature by building upon two existing methods for performing Bayesian model comparison, analyzing them theoretically and empirically, and comparing their performance. Chapter 2 presents two parallel algorithms based on nested sampling and shows results for the new methods tested on several example problems. Chapter 3 presents three parallel algorithms based on thermodynamic integration with adaptive annealing and shows results for these methods tested on the same examples as in Chapter 2. Chapter 4 presents theoretical and empirical time complexity analyses for the proposed algorithms, and it examines the empirical precision of the log-evidence estimate across ranges of method parameter values. Chapter 5 concludes the dissertation.

This work will most immediately benefit investigators in various fields that make use of Bayesian model comparison on a regular basis. Most contemporary improvements in computing performance per watt come in the form of increasingly parallel architectures rather than serial ones. Hence, model comparison algorithms that can take advantage of parallel computing hardware are better equipped to take advantage of new platforms as they become available.

CHAPTER 2

NESTED SAMPLING-BASED METHODS

This chapter describes parallelized methods I have developed based on nested sampling to perform model comparison. This chapter is largely adapted from [Henderson et al., 2017] and, to a lesser extent, [Henderson and Goggans, 2014].

Nested sampling [Skilling, 2004b, 2006] is a numerical technique for evaluating model evidence integrals. These integrals often present major challenges: they can be high-dimensional, a significant portion of their mass is often concentrated in a very small area of the parameter space, and they can be multi-modal. Nested sampling's main strength lies in mitigating these challenges.

2.1 Nested Sampling Derivation

Burrows [1980] introduced a method for estimating multi-dimensional integrals using Lebesgue integration and an estimate of the Lebesgue measure. Nested sampling uses a similar reparameterization to compute evidence integrals.

Let us introduce some abbreviations to make the following derivation more concise: denote the evidence $\mathrm{p}(\boldsymbol{D}|\boldsymbol{M}, I)$ as $\mathcal{Z}$, the likelihood $\mathrm{p}(\boldsymbol{D}|\boldsymbol{\Theta}, \boldsymbol{M}, I)$ as $L(\boldsymbol{\Theta})$, and the prior $\mathrm{p}(\boldsymbol{\Theta}|\boldsymbol{M}, I)$ as $\pi(\boldsymbol{\Theta})$. With these abbreviations, the evidence integral (1.29) becomes

$$\mathcal{Z} = \int \pi(\boldsymbol{\Theta}) L(\boldsymbol{\Theta}) \, \mathrm{d}\boldsymbol{\Theta}. \tag{2.1}$$

Introduce dummy variable $\mathcal{L}$ and let

$$L(\boldsymbol{\Theta}) = \int_0^{L(\boldsymbol{\Theta})} \mathrm{d}\mathcal{L}. \tag{2.2}$$

(2.1) becomes

$$\mathcal{Z} = \int \pi(\boldsymbol{\Theta}) \left[\int_0^{L(\boldsymbol{\Theta})} \mathrm{d}\mathcal{L} \right] \mathrm{d}\boldsymbol{\Theta}. \tag{2.3}$$

Now, bring $\pi(\boldsymbol{\Theta})$ inside the integral over $\mathcal{L}$, and reverse the order of integration, observing that $L(\boldsymbol{\Theta}) > \mathcal{L}$ and $0 \leq \mathcal{L} < \infty$,

$$\mathcal{Z} = \int_0^{\infty} \left[\int_{\{\boldsymbol{\Theta}: L(\boldsymbol{\Theta}) > \mathcal{L}\}} \pi(\boldsymbol{\Theta}) \, \mathrm{d}\boldsymbol{\Theta} \right] \mathrm{d}\mathcal{L}. \tag{2.4}$$

Define the following non-increasing function,

$$X(\mathcal{L}) = \int_{\{\boldsymbol{\Theta}: L(\boldsymbol{\Theta}) > \mathcal{L}\}} \pi(\boldsymbol{\Theta}) \, \mathrm{d}\boldsymbol{\Theta}. \tag{2.5}$$

Hence,

$$\mathcal{Z} = \int_0^{\infty} X(\mathcal{L}) \, \mathrm{d}\mathcal{L}. \tag{2.6}$$

We observe that $X(0) = 1$ and $\lim_{\mathcal{L} \to \infty} X(\mathcal{L}) = 0$ and that if we take $\mathcal{L}(X)$ as a Lebesgue measure of X, the Lebesgue integral for (2.6) is

$$\mathcal{Z} = \int_0^1 \mathcal{L}(X) \, \mathrm{d}X. \tag{2.7}$$

In (2.5), our dummy variable $\mathcal{L}$ places a lower limit on the likelihood function. Therefore, it is natural to interpret it as a likelihood constraint. $X(\mathcal{L})$ is the proportion of the prior contained within the likelihood constraint $\mathcal{L}$, and it is hence called the prior mass.

The integral (2.5) is just as difficult to evaluate as (2.1); however, in most cases, X can be estimated within an acceptable amount of error, so we can compute an estimate of (2.4) using quadrature. The original nested sampling algorithm [Skilling, 2006] proceeds as follows. The initial likelihood threshold is set to $\mathcal{L} = 0$. N so-called live samples are

drawn from the prior distribution $\pi(\mathbf{\Theta})$. The sample with the lowest likelihood value is removed from the set of live samples, and its likelihood is chosen as the next likelihood threshold $\mathcal{L}$. A new sample is drawn from $\pi(\mathbf{\Theta})$, constrained by the new likelihood threshold $\mathcal{L}$. This process is repeated until enough likelihood-prior mass pairs have been generated to adequately estimate the evidence.

Because the live samples at any step in the nested sampling process are distributed according to the constrained prior distribution, their prior mass values are uniformly distributed on $[0, X(\mathcal{L})]$. Let us look more closely at the first step. At the first step, the prior mass of the live samples is uniformly distributed on $[0, 1]$. According to the order statistics of the uniform distribution, the prior mass of the sample with the largest prior mass X_1 (which corresponds to the lowest likelihood $\mathcal{L}_1$) is distributed according to a beta distribution with parameters N and 1. The ratio of the prior mass at $\mathcal{L}_1$ (X_1) to the prior mass at $\mathcal{L} = 0$ ($X = 1$) is

$$t_1 = \frac{X_1}{1} = X_1. \tag{2.8}$$

The shrinkage at step i of the nested sampling process maintains the same beta distribution, so we can write

$$t_i \sim \text{Beta}(N, 1), \tag{2.9}$$

and the prior mass at step i can be written

$$X_i = \prod_{k=1}^{i} t_k. \tag{2.10}$$

The most useful estimate of the shrinkage at each step turns out to be the log-geometric mean,

$$\text{E}(\log t_i) = -1/N. \tag{2.11}$$

Hence, at step i, a useful estimate of the prior mass is

$$X_i \approx \exp(-i/N). \tag{2.12}$$

With this, we have all the ingredients to build our quadrature approximation of the evidence,

$$Z \approx \sum_{i=1}^{m} (X_{i-1} - X_i)\, \mathcal{L}_i, \tag{2.13}$$

using the likelihood function to find the exact likelihood of each least-likelihood sample and (2.12) to estimate each prior mass value.

A reference implementation of nested sampling in pseudo-code is shown in Algorithm 2.1.

2.1.1 Implementation considerations

There are several details of the original nested sampling algorithm that are left up to the user to decide how to implement. These details include how to set the number of live samples N, how to decide when enough likelihood-prior mass sample pairs have been generated to obtain an adequate estimate of the evidence, and how to efficiently sample the prior distribution constrained by $\mathcal{L}$. Regarding the choice of N, Skilling [2006] derives the upper bound on the error in the evidence as $\exp\left(\sqrt{H/N}\right)$, where H is the negative relative entropy of the posterior distribution with respect to the prior distribution. (Note that this bound assumes that quadrature error is negligible and that the constrained prior is accurately sampled at each step.) This straightforward relationship between the expected error in the evidence estimate and N gives the user an appropriate place to start in choosing a value.

According to (2.11), each step in the nested sampling process shrinks the volume being explored by $\exp(-1/N)$. Under perfect circumstances, it would therefore take NH steps to compress by $\exp(-H)$ and reach the bulk of the posterior distribution. Recognizing

Algorithm 2.1 Nested sampling

procedure NestedSampling($P, N, MIN, MAX, data$)
  $logZ \leftarrow -\infty$
  $H \leftarrow 0.0$
  $logwidth \leftarrow \log(1 - \exp(-1/N))$
  for $i \leftarrow 1, N$ **do**
    for $j \leftarrow 1, P$ **do**
      $\theta_{i,j} \leftarrow \text{uniform}(0, 1)$
    end for
    $logL_i \leftarrow \text{LogLikelihood}(\theta_i, data)$
  end for
  $iterate \leftarrow 1$
  $count \leftarrow 0$
  while $iterate = 1$ **do**
    $r \leftarrow \arg\min_i(logL_i)$
    $minlogL \leftarrow logL_r$
    $logWt \leftarrow minlogL + logwidth$
    $logZ_{old} \leftarrow logZ$
    $logZ \leftarrow \log(\exp(logZ) + \exp(logWt))$
    $H \leftarrow \exp(logWt - logZ)minlogL + \exp(logZ_{old} - logZ)(H + logZ_{old})$
    $q \leftarrow \lfloor (N + 1)uniform(0, 1) \rfloor$
    $\theta_r, logL_r \leftarrow \text{MCMC}(\theta_q, data)$
    $logwidth \leftarrow logwidth - 1/N$
    $count \leftarrow count + 1$
    if $count > MIN$ and $count > 3HN$ **then**
      $iterate \leftarrow 0$
    else if $count \geq MAX$ **then**
      $iterate \leftarrow 0$
    end if
  end while
  return $logZ, H$
end procedure

that circumstances are generally never perfect, it is advisable to take at least $2NH$ steps to make sure an adequate estimate of the evidence is reached. In practice, up to $3NH$ samples may be required; some trial and error with the application is generally necessary to determine the best value.

How best to sample the constrained prior distribution is a matter of debate. It is usually impossible to directly sample the constrained prior, and rejection sampling becomes increasingly inefficient as the prior mass decreases. MCMC can be used to good effect here, but there are some challenges unique to sampling this type of distribution, including choosing a starting point for the Markov chain, how best to deal with the hard edges of the distribution, and how to handle multi-modal likelihood functions. Investigators have devoted significant effort to determining the best approach here. Most prominent among these efforts are MultiNest by Feroz et al. [2009], Diffusive Nested Sampling by Brewer et al. [2011], and PolyChord by Handley et al. [2015]. Skilling [2012] himself tried to resolve the issue of constrained prior sampling with Galilean Monte Carlo, but this method presents further difficult choices in implementation, and as such, it finds little application.

2.2 Parallel nested sampling

As in other numerical integration techniques, there is an antagonistic relationship between precision and speed in nested sampling. Precision in the log-evidence estimate increases with the square root of N, whereas the number of necessary likelihood thresholds increases with N. In this section we discuss our new method for parallelizing nested sampling (subsection 2.2.1) by combining the results (i.e., the discarded samples) of several nested sampling runs. We also contrast this new technique with the above-mentioned method for discarding and replacing multiple live samples at once (subsection 2.2.2), originally developed by Burkoff et al. [2012] and refined in [Henderson and Goggans, 2014].

2.2.1 Combining Independent Chains

Our new method aims to side-step the antagonism between precision and speed by combining the results (discarded samples) of several nested sampling runs, sorting the combined samples by likelihood, estimating the new shrinkage between consecutive pairs of samples, and computing a new evidence estimate. To do this it is necessary to know the distribution of the shrinkage in a combined and re-sorted set of discarded samples. Lemmas 1 and 2 and Theorem 1 establish this distribution.

Lemma 1. The negative log of the prior mass X_i for the ith sample discarded from a nested sampling run using N live samples has an Erlang density,

$$f_{-\log X_i}(x) = \frac{N^i x^{i-1} \exp(-Nx)}{(i-1)!}, \tag{2.14}$$

with shape parameter i and rate parameter N.

Proof. We prove Lemma 1 using the method of induction. Begin with the density of the shrinkage between the $(i+1)$th and ith prior mass values, (2.9) and write it in functional form,

$$f_{t_i}(t) = Nt^{N-1}. \tag{2.15}$$

From (2.10), we can write the prior mass of the second sample as

$$X_2 = t_1 t_2 \tag{2.16}$$

so that, by taking the negative logarithm of both sides,

$$-\log X_2 = (-\log t_1) + (-\log t_2). \tag{2.17}$$

The density of the 1-to-1 function $Y = g(X)$ of a continuous numerical proposition X is

given in probability texts such as [Ross, 2010] as

$$f_Y(y) = \begin{cases} f_X(h(y)) \left| \frac{dh}{dy} \right|, & h(y) \in \tilde{X} \\ 0, & \text{otherwise} \end{cases}, \tag{2.18}$$

where $h(Y)$ is the inverse of $g(X)$, and $\tilde{X}$ is the support of $f_X(x)$. Here, $Y = g(t_i) = -\log t_i$ and $h(Y) = \exp(-Y)$. By applying (2.18), we have

$$f_{-\log t_i}(y) = f_{t_i}(\exp(-y)) \left| \frac{d}{dy} \exp(-y) \right| = N \exp(-Ny). \tag{2.19}$$

Equation (2.19) shows that $-\log t_i$ has an exponential density with mean $1/N$. The sum of two continuous numerical propositions has a density given by the convolution of the densities of the two propositions [Ross, 2010]:

$$f_{-\log X_2}(x) = f_{-\log t_1 - \log t_2}(x) = \int N \exp(-Nk) N \exp[-(x-k)N]\, dk. \tag{2.20}$$

An exponential density has support on the interval $[0, \infty)$, so we have the following restrictions: $0 \leq k < \infty$ and $0 \leq x - k < \infty$, or in other words

$$0 \leq k \leq x. \tag{2.21}$$

The convolution integral becomes

$$f_{-\log X_2}(x) = N^2 \exp(-Nx) \int_0^x db = N^2 x \exp(-Nx). \tag{2.22}$$

Equation (2.22) establishes the base case ($k = 1$) for the method of induction.

Assume that the expression for $k = i-1$ is given by

$$f_{-\log X_{i-1}}(x) = \frac{N^{i-1}x^{i-2}\exp(-Nx)}{(i-2)!}. \tag{2.23}$$

By using (2.20) and the previously mentioned limits on distribution support, we have

$$f_{-\log X_i}(x) = \int_0^x \frac{N^{i-1}k^{i-2}\exp(-Nk)}{(i-2)!} N\exp[-(x-k)N]\,dk, \tag{2.24}$$

$$= \frac{N^i x^{i-1}\exp(-Nx)}{(i-1)!}. \tag{2.25}$$

Equation (2.25) specifies a gamma density with shape parameter i and rate parameter N. As i is an integer, (2.25) is specifically an Erlang density Weisstein [2016]. □

Lemma 2. Let $\{M_1(t);\, 0 < t < 1\}$ and $\{M_2(t);\, 0 < t < 1\}$ be two independent processes, $0 < t < 1$, and let the following conditions hold:

$$M_i(t) \geq 0 \tag{2.26}$$

$$M_i(t) \in \mathbb{Z} \tag{2.27}$$

$$t_2 \geq t_1 \rightarrow M_i(t_2) \leq M_i(t_1). \tag{2.28}$$

Let $\mathcal{H}\{\cdot\}$ be a transformation that executes the substitution $t = \exp(-s)$, such that $N_i(s) = \mathcal{H}\{M_i(t)\}$, and $N_i(s)$ is a Poisson process. Let the sum of two processes, e.g. $M_\Sigma(t) = M_1(t) + M_2(t)$, represent a single process with events that consist of the combination of events from the individual processes. Then the transformation $\mathcal{H}\{\cdot\}$ is linear in addition, i.e.

$$\mathcal{H}\{M_1(t) + M_2(t)\} = \mathcal{H}\{M_1(t)\} + \mathcal{H}\{M_2(t)\} = N_1(s) + N_2(s). \tag{2.29}$$

Proof. $M_1(t)$ and $M_2(t)$ are discrete random variables parameterized by t. A necessary

condition for (2.29) to be satisfied is therefore that the p.m.f. of the RV on the left is equivalent to the p.m.f. of the RV on the right.

Let $\{N_1(s);\, s > 0\}$ and $\{N_2(s);\, s > 0\}$ be Poisson counting processes with rates λ_1 and λ_2. Let $N(s) = N_1(s) + N_2(s)$. The probability mass functions for $N_1(s)$ and $N_2(s)$ are given as

$$p_{N_1(s)}(n_1) = \frac{(\lambda_1 s)^{n_1} \exp(-\lambda_1 s)}{n_1!} \tag{2.30}$$

$$p_{N_2(s)}(n_2) = \frac{(\lambda_2 s)^{n_2} \exp(-\lambda_2 s)}{n_2!}. \tag{2.31}$$

The distribution of $N(s)$, which comprises the sum of $N_1(s)$ and $N_2(s)$, is given as a convolution of the distributions for $N_1(s)$ and $N_2(s)$:

$$p_{N(s)}(n) = \sum_{k=-\infty}^{\infty} p_{N_1(s)}(k) p_{N_2(s)}(n-k). \tag{2.32}$$

These are counting processes, which are zero-valued unless $k \geq 0$ and $n - k \geq 0$. Thus, $0 \leq k \leq n$, and the convolution sum (2.32) becomes

$$p_{N(s)}(n) = \sum_{k=0}^{n} p_{N_1(s)}(k) p_{N_2(s)}(n-k) \tag{2.33}$$

$$= \sum_{k=0}^{n} \frac{(\lambda_1 s)^k \exp(-\lambda_1 s)}{k!} \frac{(\lambda_2 s)^{n-k} \exp(-\lambda_2 s)}{(n-k)!} \tag{2.34}$$

$$= \sum_{k=0}^{n} \frac{(\lambda_1/\lambda_2)^k (\lambda_2 s)^n \exp[-(\lambda_1 + \lambda_2) s]}{k!(n-k)!} \tag{2.35}$$

$$= (\lambda_2 s)^n \exp[-(\lambda_1 + \lambda_2) s] \sum_{k=0}^{n} \frac{(\lambda_1/\lambda_2)^k}{k!(n-k)!} \tag{2.36}$$

$$= (\lambda_2 s)^n \exp[-(\lambda_1 + \lambda_2) s] \frac{[(\lambda_1 + \lambda_2)/\lambda_2]^n}{n!} \tag{2.37}$$

$$= \frac{[s(\lambda_1 + \lambda_2)]^n \exp[-(\lambda_1 + \lambda_2) s]}{n!}. \tag{2.38}$$

This result establishes the right-hand side of (2.29).

Now carry out the inverse transform $\mathcal{H}^{-1}\{\cdot\}$ on (2.30) and (2.31):

$$p_{M_1(t)}(m_1) = \frac{(-\lambda_1 \log(t))^{m_1} \exp(\lambda_1 \log(t))}{m_1!} = \frac{(-\lambda_1 \log(t))^{m_1} t^{\lambda_1}}{m_1!} \tag{2.39}$$

$$p_{M_2(t)}(m_2) = \frac{(-\lambda_2 \log(t))^{m_2} \exp(\lambda_2 \log(t))}{m_2!} = \frac{(-\lambda_2 \log(t))^{m_2} t^{\lambda_2}}{m_2!}. \tag{2.40}$$

Let $M(t) = M_1(t) + M_2(t)$, and find the p.m.f. for $M(t)$ as in (2.32). The result is

$$p_{M(t)}(m) = \frac{(-1)^m (\lambda_1 + \lambda_2)^m (\log t)^m t^{\lambda_1+\lambda_2}}{m!}. \tag{2.41}$$

Carry out the transform

$$p_{\mathcal{H}\{M(t)\}}(m) = p_{N(s)}(n) = \frac{[s(\lambda_1 + \lambda_2)]^n \exp[-(\lambda_1 + \lambda_2)s]}{n!} \tag{2.42}$$

(2.38) and (2.42) are identical, so we have shown that

$$p_{\mathcal{H}\{M_1(t)+M_2(t)\}}(m) = p_{\mathcal{H}\{M_1(t)\}+\mathcal{H}\{M_2(t)\}}(m). \tag{2.43}$$

As $M(t)$ and $N(s)$ represent random processes, their p.m.f.s are actually probability mass *functionals*, not functions. Let $f(t) = t^{\lambda_1+\lambda_2}$ and $g(s) = (\lambda_1 + \lambda_2)s$. We can express these functionals as

$$p_{M(t)}(m) = F_1[f(t)] = \frac{(-1)^m (\log f(t))^m f(t)}{m!} \tag{2.44}$$

$$p_{N(s)}(n) = F_2[g(t)] = \frac{[g(s)]^n \exp[-g(s)]}{n!}. \tag{2.45}$$

The mean and autocorrelation functions of these mass functionals can depend *only* on the functions $f(t)$ and $g(s)$. Since $f(t)$ is the same on both sides of (2.29), as we have shown in (2.38) and (2.42), it follows that the mean and autocorrelation functions are also equivalent,

and that:

$$\mathcal{H}\left\{M_1(t) + M_2(t)\right\} = \mathcal{H}\left\{M_1(t)\right\} + \mathcal{H}\left\{M_2(t)\right\}. \tag{2.46}$$

We can therefore say that $\mathcal{H}\{\cdot\}$ is linear in addition. □

Theorem 1. The shrinkage s_i between consecutive prior mass values Y_{i+1} and Y_i in a set of discarded samples combined from M independent nested sampling runs, each using N_j, $j = 1, \cdots, M$ live samples, has a beta density

$$f_{s_i}(s) = \left(\sum_{j=1}^{M} N_j\right) s^{\left(\sum_{j=1}^{M} N_j\right)-1} \tag{2.47}$$

with shape parameters $\sum_{j=1}^{M} N_j$ and 1.

Proof. The wait time for the ith event in a Poisson process is distributed as an Erlang density (which is a special case of the gamma density with an integer shape parameter), [Taylor and Karlin, 1998, p. 291] as shown in (2.25). This fact and Lemma 1 show that the variable $-\log X_i$ for a set of samples discarded from a single run of nested sampling follows a Poisson process. We can therefore apply the properties of Poisson processes to see what happens when we combine sets of discarded samples from multiple independent runs of nested sampling.

Let Y_i be the prior mass associated with the ith sample in a set of discarded samples combined from M independent nested sampling runs, each of which uses N_j, $j = 1, \cdots, M$ live samples. Let $Q = \sum_{j=1}^{M} N_j$. If M separate and independent Poisson processes with intensities N_j, $j = 1, \cdots, M$ are combined, the result is a single Poisson process with intensity $\sum_{j=1}^{M} N_j$ [Tijms, 2003, p. 6]. It follows that $-\log Y_i$ is distributed as

$$f_{-\log Y_i}(y) = \frac{Q^i y^{i-1} \exp(-Qy)}{(i-1)!}, \tag{2.48}$$

which is an Erlang density with shape parameter i and rate parameter Q.

It follows from the definition of the shrinkage that $-\log Y_i = \sum_{j=1}^{i} -\log s_j$. The sum

of propositions with identical densities has a gamma density if and only if the propositions being added also have gamma densities. In this case where the shape parameter is 1, the densities of $-\log s_i$ are also exponential,

$$f_{-\log s_i}(s) = Q \exp(-Qs). \tag{2.49}$$

The transform in (2.19) leads to

$$f_{s_i}(s) = Q s^{Q-1}, \tag{2.50}$$

which is the beta distribution with parameters $Q = \sum_{j=1}^{M} N_j$ and 1. Finally, Lemma 2 shows that the transformation from a process with wait time $-\log s_i$ to a process with wait time s_i is a linear transformation, and our result follows. □

Theorem 1 demonstrates that using a combined set of discarded samples from M independent nested sampling processes which use N live samples each, the evidence estimate can be computed as if one nested sampling process had produced the samples, using MN live samples.

2.2.1.1 Numerical example

We now give a numerical example that demonstrates the validity of the result in theorem 1.

Example. Let $M = 32$. Generate M sets of $10,000$ shrinkage samples from Beta$(100, 1)$ (i.e., $N = 100$). Use (2.10) to compute the associated prior mass for each sample in each set. Combine the independent sets of samples, then sort the combined sample by prior mass. [1] Compute the actual shrinkage between each pair of consecutive samples. Compare the cumulative distribution function (CDF) of the combined shrinkage samples with the CDFs of Beta$(N, 1)$ and Beta$(NM, 1)$. The CDF of the samples should closely match the CDF of Beta$(NM, 1)$.

[1] Prior mass is a monotonic function of the likelihood constraint, so that sorting by prior mass is equivalent to sorting by likelihood.

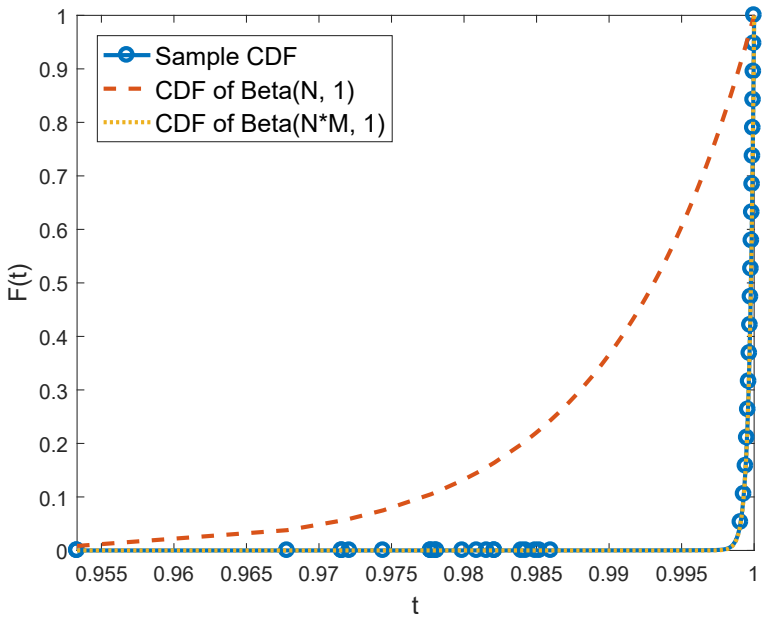


Figure 2.1: CDFs for the combined and sampled shrinkage, Beta$(N, 1)$, and Beta$(NM, 1)$

Result. Execution of this procedure yielded the empirical CDF shown in Figure 2.1. The CDFs for Beta$(N, 1)$ and Beta$(NM, 1)$ are also shown in Figure 2.1. The empirical CDF closely matches the CDF for Beta$(NM, 1)$, confirming the result in Theorem 1.

2.2.2 Discarding and Replacing Several Samples at Once

Here we describe the parallel nested sampling technique developed by Burkoff et al. [2012] and subsequently refined by Henderson and Goggans [2014]. The main idea behind this method for parallelizing nested sampling is that, for a single nested sampling process with N_1 live samples, by discarding and replacing $R \ll N_1$ samples at once we need only HN_1/R steps to reach the desired convergence in the log-evidence estimate. The variance of the shrinkage distribution also increases linearly with R [Skilling, 2012], so if we wish to maintain the same precision in the log-evidence estimate then we must also scale the number of live samples as $N_R = \sqrt{R}N_1$. This scaling implies that $HN_1/\sqrt{R}$ steps are required to reach convergence, so that our speed-up factor is approximately $\sqrt{R}$. A reference implementation of multiple-replacement nested sampling is shown in Algorithm 2.2.

Algorithm 2.2 Multiple-replacement nested sampling

procedure NESTEDSAMPLINGMR($P, N, R, MIN, MAX, data$)
  $logZ \leftarrow -\infty$
  $H \leftarrow 0.0$
  $logwidth \leftarrow \log(1 - \exp(-R/N))$
  for $i \leftarrow 1, N$ **do**
    for $j \leftarrow 1, P$ **do**
      $\theta_{i,j} \leftarrow \text{uniform}(0, 1)$
    end for
    $logL_i \leftarrow \text{LOGLIKELIHOOD}(\theta_i, data)$
  end for
  $iterate \leftarrow 1$
  $count \leftarrow 0$
  while $iterate = 1$ **do**
    Sort θ and $logL$ by $logL$ in ascending order
    $logWt \leftarrow logL_R + logwidth$
    $logZ_{old} \leftarrow logZ$
    $logZ \leftarrow \log(\exp(logZ) + \exp(logWt))$
    $H \leftarrow \exp(logWt - logZ) minlogL + \exp(logZ_{old} - logZ)(H + logZ_{old})$
    $q \leftarrow$ Choose R random integers on $[R + 1, N]$
    for $r \leftarrow 1, R$ **do**
      $\theta_r, logL_r \leftarrow \text{MCMC}(\theta_{q_r}, data)$
    end for
    $logwidth \leftarrow logwidth - R/N$
    $count \leftarrow count + 1$
    if $count > MIN$ and $count > 3HN/R$ **then**
      $iterate \leftarrow 0$
    else if $count \geq MAX$ **then**
      $iterate \leftarrow 0$
    end if
  end while
  return $logZ, H$
end procedure

Algorithm 2.3 Combined-chain nested sampling

procedure NESTEDSAMPLINGCC($P, N, M, MIN, MAX, data$)
    $samples \leftarrow$ empty list
    for $i \leftarrow 1, M$ **do**
        APPEND($samples$, NESTEDSAMPLING($P, N, MIN, MAX, data$))
    end for
    Sort $samples$ by $logL$
    $logwidth \leftarrow \log(1 - \exp(-1/(MN)))$
    $logZ \leftarrow -\infty$
    $H \leftarrow 0.0$
    for $i \leftarrow 1,$ LENGTH($samples$) **do**
        $logWt \leftarrow samples[i].logL + logwidth$
        $logZ_{old} \leftarrow logZ$
        $logZ \leftarrow \log(\exp(logZ) + \exp(logWt))$
        $H \leftarrow \exp(logWt - logZ) samples[i].logL + \exp(logZ_{old} - logZ)(H + logZ_{old})$
    end for
    return $logZ, H$
end procedure

2.2.3 Implementation

Implementation of the method described in section 2.2 is flexible, and depends on the user's specific needs. The main idea is that two or more independent runs of nested sampling are conducted with identical numbers of live samples, and the results are then combined and used to generate an estimate of the evidence.

The combination of results for multiple runs proceeds as follows. Let each sample be represented by a data structure with three fields: parameter array, log-likelihood, and log-weight. The samples from each run are combined into one large array, the associated log-weights are discarded, and the samples are sorted by log-likelihood. The log-weight is then re-estimated for each sample using the new ordering and the combined number of live samples, according to the shrinkage distribution given in Theorem 1. The log-evidence is then estimated according to (2.13) using the new log-weights. A reference implementation of combined-chain nested sampling following this procedure is shown in Algorithm 2.3.

The nested sampling runs can be implemented concurrently or sequentially. Here are some situations that might plausibly arise:

- If you know the value of N necessary to give the precision you require, then instead of performing one run with N live samples, perform M runs concurrently on multiple processor cores or supercomputer nodes, each run using $\lceil N/M \rceil$ live samples.

- If you have an evidence estimate but it is not precise enough, perform several more nested sampling runs incorporating the data, then combine the old and new results to obtain a more precise estimate.

- If you suspect that the distribution you are exploring is highly multi-modal, perform many independent nested sampling runs using a relatively small N so as to increase the likelihood that each mode is well-explored.

2.2.3.1 Speed-up

In the first of these three cases the speed-up may be substantial, theoretically by up to a factor of M, but this is limited by several factors. Each nested sampling run involves no communication with the other runs. The speed-up is therefore limited by the total number of live samples used, by the time needed to start each independent process, and by the time needed to combine and process the final results.

Each process must use at least one live sample. If N total live samples are used then not more than N processing units can be used to divide the work.

Whatever the specific architecture used to parallelize the computation, a certain amount of time and computational effort will be needed to copy the necessary data and instructions to the individual processing units before beginning each run. Further time is involved when each processing unit is finished and communicates its results back to the central controlling implementer. Also, the combining of the sets of discarded samples, recomputing of the sample weights, and computing the final evidence takes further time.

Relative to the older method of parallelizing nested sampling by discarding and replacing multiples samples at once, the theoretical speed-up, given M workers, is by a factor of $\sqrt{M}$. Thus, our new method requires $1/\sqrt{M}$ as many likelihood evaluations per worker

compared with the older method. The newer method should therefore be preferred over the older method in all cases, unless an external factor (such as the MCMC method used to replace samples) points to use of the older method.

2.3 Examples

In this section we demonstrate the performance of combined-chain nested sampling using three problems as examples. These include an artificial, highly multi-modal likelihood function, a simulated spectrum analysis problem, and a twin Gaussian shell problem in 20 and 30 dimensions. For these examples, we use nested sampling as described in sections 2.1 and 2.2. Each example uses a different number of live samples N and a different number of independent runs M.

The method used to replace the discarded sample at each likelihood constraint is slightly different in each example. In both examples we use some form of Metropolis MCMC, which takes a sample at random from the surviving collection of live samples and moves it randomly around the likelihood-constrained parameter space. The final location is used as the new live sample. Both examples use uniform priors, and the Metropolis acceptance criterion is then simple: if at any time the sample lands outside of the likelihood constraint, it is rejected; if it lands inside the likelihood constraint, it is accepted.

In the following examples, constrained prior exploration differs in how the samples are moved throughout the parameter space. For the eggcrate example and the twin Gaussian shell example (section 2.3.1 and 2.3.3), the moves are drawn from a standard normal distribution scaled by a factor that is adjusted at each step. Both parameters are changed simultaneously, and, at the end of each step, the step size scaling factor is increased or decreased so as to maintain a roughly even acceptance ratio.

For the multiple stationary frequency example (section 2.3.2), a Gaussian step is also used, but each parameter is varied individually. At each step of the exploration procedure a random ordering of the parameters is chosen, and the sample is moved in one dimension

at a time. The acceptance rate is monitored separately for each parameter, and a scaling factor corresponding to each parameter is updated at the end of each MCMC step in order to maintain a roughly even acceptance ratio.

2.3.1 Eggcrate Likelihood

The first example is the "eggcrate" toy problem from [Feroz et al., 2009]. The posterior density in this problem is highly multimodal, so that a large number of live samples is necessary to estimate the evidence accurately and to sample the entire density. Our results for this example show that combining many independent nested sampling chains with few live samples gives a result equivalent to using one chain with many live samples in the presence of many modes.

The eggcrate function has two independent parameters, with joint prior

$$\pi(\boldsymbol{\Theta}) = \left(\frac{1}{10\pi}\right)^2 \mathbb{1}_{[0,10\pi]}(\Theta_1)\mathbb{1}_{[0,10\pi]}(\Theta_2). \tag{2.51}$$

The likelihood function is

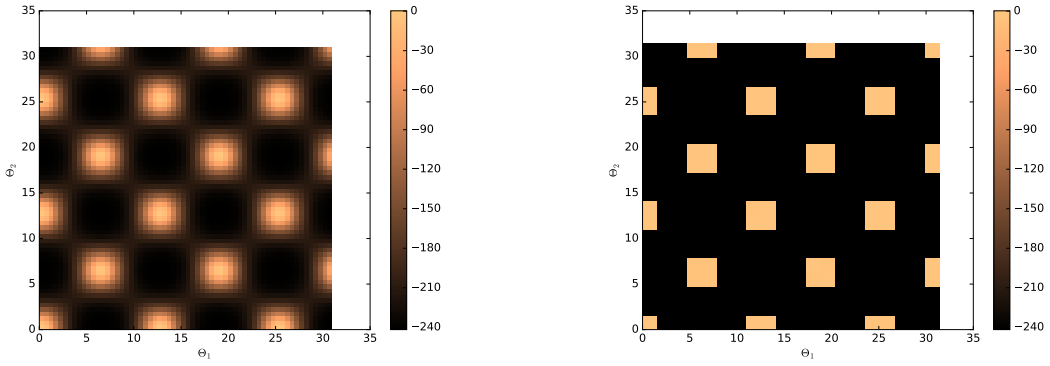
$$\mathcal{L}(\boldsymbol{\Theta}) = \exp\left\{\left[2 + \cos\left(\frac{\Theta_1}{2}\right)\cos\left(\frac{\Theta_2}{2}\right)\right]^5\right\}. \tag{2.52}$$

In this case the log-likelihood is more useful for visualization and the numerical dynamic range:

$$\log\mathcal{L}(\boldsymbol{\Theta}) = \left[2 + \cos\left(\frac{\Theta_1}{2}\right)\cos\left(\frac{\Theta_2}{2}\right)\right]^5. \tag{2.53}$$

Feroz et al. [2009] provide an evidence value of $\log\mathcal{Z} = 235.88$ using numerical integration over a fine grid. Upon applying Bayes' theorem, the posterior distribution for the parameters $\boldsymbol{\Theta}$ is

$$\mathcal{P}(\boldsymbol{\Theta}) = \frac{\exp\left\{\left[2 + \cos\left(\frac{\Theta_1}{2}\right)\cos\left(\frac{\Theta_2}{2}\right)\right]^5\right\}\left(\frac{1}{10\pi}\right)^2 \mathbb{1}_{[0,10\pi]}(\Theta_1)\mathbb{1}_{[0,10\pi]}(\Theta_2)}{\exp(235.88)}. \tag{2.54}$$



(a) Log-posterior for the eggcrate problem (log of (2.54))

(b) Log pseudo-color plot of normalized histogram of re-sampled nested sampling results, 20x20 bin grid

Figure 2.2: Eggcrate figures

2.3.1.1 Results

The results shown here were achieved using $N = 16$ live samples in each of $M = 20$ independent nested sampling runs. The uncertainty in the log-evidence estimate was estimated by performing 20 separate runs of the configuration with $N = 16$, $M = 20$. The estimated log-evidence value is $\log \mathcal{Z} = 235.84 \pm 0.1616$. This estimate is well within a single standard deviation of the log-evidence estimate of 235.88 given in [Feroz et al., 2009]. Figure 2.2a shows the log-posterior for the eggcrate function, and Figure 2.2b shows a normalized log-histogram of the samples obtained using nested sampling, resampled using importance sampling as described by Goggans and Chi [2004]. The normalized log-histogram in Figure 2.2b shows that each mode of the posterior was sampled well.

2.3.2 Detection of Multiple Stationary Frequencies

In the second example, we want to estimate the number of stationary frequencies present in a signal as well as the value of each frequency. This problem is similar to the problem of multiple stationary frequency estimation in [Bretthorst, 1988, Chapter 6], with the additional task of determining the number of stationary frequencies present. This example demonstrates the value of the present parallel nested sampling method. Differences

among log-evidence values for models containing either the most probable number of frequencies or more tend to be small, meaning that a precise estimate of these log-evidence values is essential to the task of determining the most probable model.

Each stationary frequency (j) in the model is determined by three parameters: the in-phase amplitude (A_j), the quadrature amplitude (B_j), and the frequency (f_j). Given J stationary frequencies, the model at time step t_i takes the following form:

$$g\left(t_i;\boldsymbol{\Theta}\right) = \sum_{j=1}^{J} A_j \cos\left(2\pi f_j t_i\right) + B_j \sin\left(2\pi f_j t_i\right), \tag{2.55}$$

where $\boldsymbol{\Theta}$ is the parameter vector

$$\boldsymbol{\Theta} = \left[A_1\, B_1\, f_1\, \cdots\, A_J\, B_J\, f_J\right]^T.$$

For the purposes of this example the noise variance used to generate the simulated data is known, and we consequently use a Gaussian likelihood function,

$$\mathcal{L}(\boldsymbol{\Theta}) = \prod_{i=1}^{I} \exp\left\{-\frac{\left[g\left(t_i;\boldsymbol{\Theta}\right) - d_i\right]^2}{2\sigma^2}\right\}, \tag{2.56}$$

for I simulated data d_i and noise variance σ^2. The log-likelihood function is then

$$\log \mathcal{L}(\boldsymbol{\Theta}) = -\sum_{i=1}^{I} \frac{\left[g\left(t_i;\boldsymbol{\Theta}\right) - d_i\right]^2}{2\sigma^2}. \tag{2.57}$$

Each model parameter is assigned a uniform prior distribution with limits as shown in Table 2.1.

Our test signal is a sum of two sinusoids, and zero-mean Gaussian noise with variance $\sigma^2 = 0.01$. This signal is sampled at randomly-spaced instants of time, in order to demonstrate that this time-domain method does not require uniform sampling to perform spectrum estimation. Bretthorst [2001] demonstrates that the Nyquist critical frequency in

Table 2.1: Prior bounds for multiple stationary frequency model parameters

	Lower bound	Upper bound
A_j	-2	2
B_j	-2	2
f_j	0 Hz	6.4 Hz

Table 2.2: Parameters used to generate simulated signal

j	A_j	B_j	f_j (Hz)
1	1.0	0.0	3.1
2	1.0	0.0	5.9

the case of nonuniform sampling is $1/2\Delta T'$, where $\Delta T'$ is the dwell time. The dwell time is not defined for arbitrary-precision time values as used in this example, so we must choose another limiting value. A more conservative limit is given by $1/10\Delta T_{\text{avg}}$, where ΔT_{avg} is the average spacing between time steps, 1/64 s. This formulation yields a prior maximum limit of 6.4 Hz, as shown in Table 2.1. The parameters used to generate the simulated data are shown in Table 2.2. The samples from the signal with noise are shown in Figure 2.3.

2.3.2.1 Results

Using the data shown in Figure 2.3, we estimate log-evidence values under the assumption that 1, 2, 3, and 4 sinusoids are present. The log-evidence is estimated using nested sampling with three sets of algorithm parameters: $N = 20$ and $M = 1$; $N = 200$ and $M = 1$; and $N = 50$ and $M = 4$. For comparison, we also include results from the old method of parallelizing nested sampling, using multiple replacement of live samples with $N = 400$ and $R = 4$. For each of the proposed models and algorithm parameter choices we conduct 20 separate nested sampling runs in order to quantify empirically the precision of the log-evidence estimate associated with each set of algorithm parameters. Box and whisker plots are shown in Figures 2.4, 2.5, and 2.6 so as to summarize the log-evidence results from

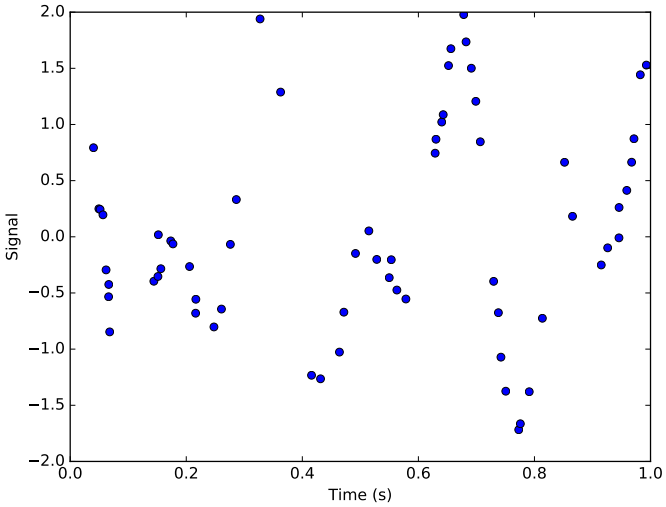


Figure 2.3: The simulated signal. The points represent the non-uniformly sampled points from the original signal corrupted by Gaussian noise.

each of these nested sampling runs. A box and whisker plot is also shown in Figure 2.7 for the log-evidence results for 20 runs each of nested sampling using the old parallel algorithm. In these box and whisker plots, the center line represents the median, the top and bottom of the box represents the third and first quartiles, the ends of the whiskers represent the maxima and minima of the observed values that fall within 1.5 times the interquartile range, and any values that fall outside that range are plotted as plus signs. The ordinate axis in each of these plots is truncated, so that the log-evidence values for each proposed model can be clearly displayed. For the 2-sinusoid model, one outlier is excluded from the plot in Figure 2.4 because it is some 10^2 standard deviations away from the mean.

Table 2.3 shows the mean and standard deviation of the log-evidence results for the 2-sinusoid model according to the trials using various parameter settings.

In the results for one run of nested sampling with $N = 20$ live samples (Figure 2.4), the log-evidence estimates for the models with 2, 3, and 4 sinusoidal components overlap to some extent. In usual practice we would only wish to perform one run of nested sampling to obtain a log-evidence estimate, but these results demonstrate that $N = 20$ does not yield sufficient precision that would let us determine the maximum a-posteriori (MAP) model

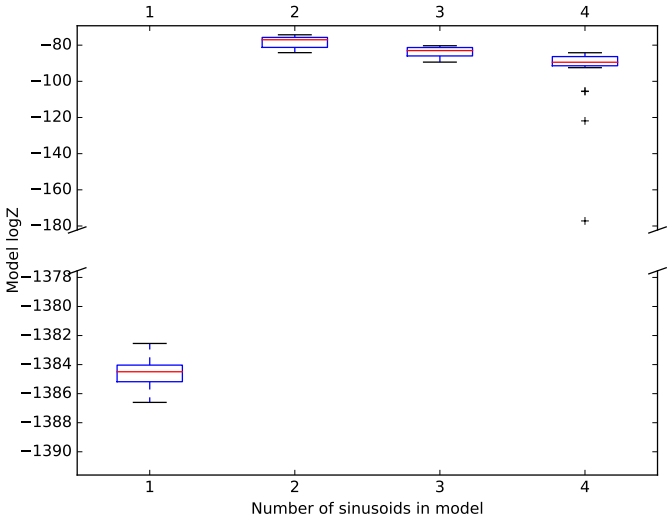


Figure 2.4: Model log-evidence, $N = 20$, $M = 1$, 20 tests each

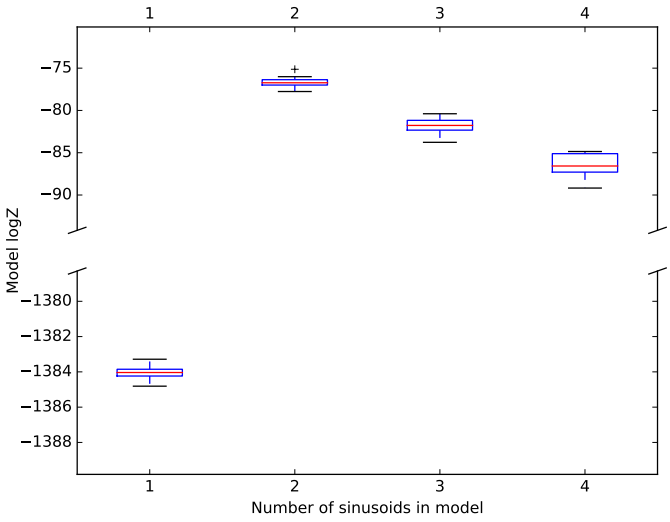


Figure 2.5: Model log-evidence, $N = 200$, $M = 1$, 20 tests each

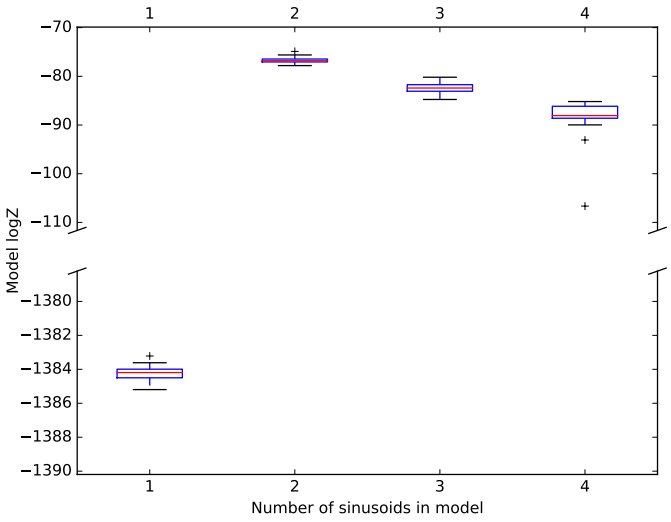


Figure 2.6: Model log-evidence, $N = 50$, $M = 4$, 20 tests each

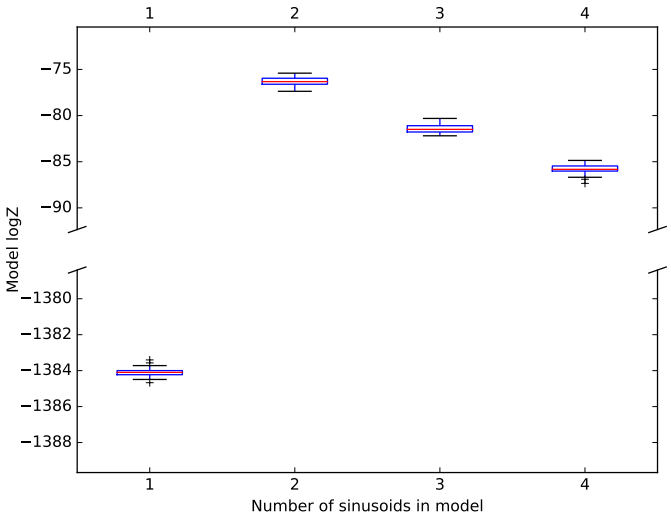


Figure 2.7: Model log-evidence, $N = 400$, $R = 4$, 20 tests each, using the old parallel nested sampling algorithm

Table 2.3: Multiple stationary frequency log-evidence for the model with 2 sinusoids, for each set of algorithm parameters. One outlier is excluded from the mean and standard deviation values for $N = 20$, $M = 1$.

Algorithm	N	M	R	Mean logZ	StDev logZ
New Parallel NS	20	1	-	-78.01	3.135
”	200	1	-	-76.68	0.5466
”	50	4	-	-76.72	0.6665
Old Parallel NS	400	-	4	-76.37	0.5512

from a single run. Figure 2.5 demonstrates that, by raising the number of live samples to $N = 200$, a level of precision is obtained that picks out model 2 as the clear MAP model, with no overlap with model 3 or model 4. Figure 2.6 shows that upon combining the results of $M = 4$ independent runs of nested sampling, each using $N = 50$ live samples, we obtain similar precision to that in the case with $M = 1$ and $N = 200$.

Table 2.3 summarizes the mean and standard deviation for the log-evidence for the 2-sinusoid model in the MSF problem. The results show that the standard deviation in the log-evidence estimates for the correct model are similar for: the $N = 200$, $M = 1$ case; the $N = 50$, $M = 4$ case; and the $N = 400$, $R = 4$ case. The standard deviation is slightly greater for the $N = 50$, $M = 4$ case, possibly indicating a less efficient likelihood-constrained prior exploration process with fewer live samples per worker. The difference is slight, however, and the speed-up apparent in the $N = 50$, $M = 4$ case supports the use of the new parallel approach in this application.

As mentioned in section 2.2, the run time of nested sampling increases linearly with the number of live samples. So, because we can obtain similar precision from 4 separate runs with $N = 50$ and 1 run with $N = 200$, we obtain a nearly four-fold speed increase by using the combined-chain nested sampling technique.

2.3.3 Twin Gaussian Shells

The final example is the twin Gaussian shell problem, also from [Feroz et al., 2009]. In [Feroz et al., 2009], the authors present results for this problem in up to 30 dimensions. Handley et al. [2015] also use this problem in 100 dimensions to test their algorithm. This problem presents a few interesting challenges to our nested sampling implementation. Because the likelihood takes the form of a thin, curved density whose mass centers on a hyper-spherical shell, exploration of the constrained prior at high likelihood values is difficult. The bimodal nature of the problem also challenges the constrained prior exploration process. Finally, the examples we explore are high-dimensional to the point that standard numerical integration techniques would be useless.

The likelihood function in the twin Gaussian shells problem takes the form,

$$\mathcal{L}(\boldsymbol{\Theta}) = \frac{1}{\sqrt{2\pi}w_1}\exp\left[-\frac{(|\boldsymbol{\Theta}-\mathbf{c}_1|-r_1)^2}{2w_1^2}\right] + \frac{1}{\sqrt{2\pi}w_2}\exp\left[-\frac{(|\boldsymbol{\Theta}-\mathbf{c}_2|-r_2)^2}{2w_2^2}\right]. \tag{2.58}$$

Following [Feroz et al., 2009], we set the parameters as follows: $w_1 = w_2 = 0.1$, $r_1 = r_2 = 2$, $\mathbf{c}_1 = [-3.5, 0, \cdots, 0]^T$, and $\mathbf{c}_2 = [3.5, 0, \cdots, 0]^T$. We use a uniform prior over the hypercube that spans $[-6, 6]$ in each dimension. Figure 2.8 shows a pseudo-color plot of a 2-dimensional twin Gaussian shell with parameters and prior range as described previously.

2.3.3.1 Results

We tested our nested sampling algorithm using this problem in 20 and 30 dimensions. The results are presented in this section.

We performed 20 tests each using $N = 200, M = 1$ and $N = 50, M = 4$ for the 20-dimensional case and $N = 300, M = 1$ and $N = 75, M = 4$ for the 30-dimensional case. The results from these tests are shown in Table 2.4. The analytic log-evidence value given in [Feroz et al., 2009] for the 20-D and 30-D twin Gaussian shell problems are -36.09 and -60.13. The results in Table 2.4 show that both algorithm settings produce results within one standard deviation of the true value for both problems. Additionally, the results for

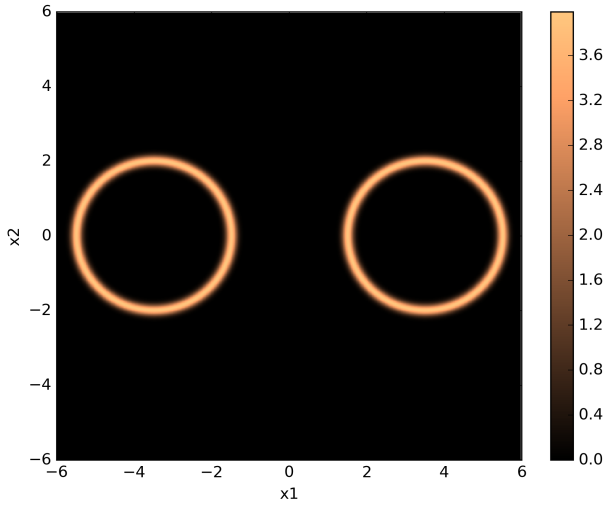


Figure 2.8: Pseudo-color plot of a 2-dimensional twin Gaussian shell with $w_1 = w_2 = 0.1$, $r_1 = r_2 = 2$, $\mathbf{c}_1 = [-3.5, 0]^T$, and $\mathbf{c}_2 = [3.5, 0]^T$. The color values correspond to likelihood values

Table 2.4: 20- and 30-dimension twin Gaussian shell log-evidence results for each algorithm setting, 20 runs each. The mean, standard deviation, and root mean squared error (RMSE) with respect to the analytic value given in [Feroz et al., 2009] are shown. The analytic log-evidence value is -36.09 for 20-D and -60.13 for 30-D.

ndim	N	M	Mean logZ	StDev logZ	RMSE
20	200	1	-36.23	0.4130	0.4349
20	50	4	-36.09	0.3341	0.3341
30	300	1	-60.22	0.4984	0.5067
30	75	4	-60.09	0.4100	0.4120

$ndim = 20$, $N = 50$, $M = 4$ have a mean value that is exactly correct up to four significant digits and have standard deviation and root mean square error values that are noticeably less than those for the $N = 200$, $M = 1$ setting. While the results for $ndim = 30$, $N = 75$, $M = 4$ do not have a mean that is exactly correct, the mean is much closer to the analytic value than that in the serial configuration, and the RMSE is again significantly less in the parallel configuration than in the serial configuration. The reason for the decrease in error for the parallel configuration versus the serial configuration could be that the parallel configuration somehow compensates for imperfect sampling of the prior distribution within each likelihood contour in the serial configuration. These results demonstrate that our algorithm can handle distributions with curving degeneracies, multi-modal distributions, and high-dimensional problems at least as well as standard (serial) nested sampling, all while providing the opportunity for significant parallel speed-up. The degree of speed-up that can be achieved when using combined-chain nested sampling is explored in Chapter 4.

2.4 Conclusion

The nested sampling algorithm, as conceived originally, is implemented in a strictly serial fashion. By combining the samples produced by multiple independent runs of nested sampling, we have developed a method for efficiently performing Bayesian model selection in a way that takes advantage of modern parallel computing architectures. We have given the mathematical foundation for estimating the evidence from these combined chains, as well as several ideas for practical implementation of the method. Three examples have demonstrated the utility and effectiveness of the technique for a variety of problem types, including a problem with a highly multi-modal distribution, a data analysis problem, and a problem with a distribution that is high-dimensional, curving, and multi-modal.

Combined-chain nested sampling is a generalized technique that is capable of providing the foundation for other more specialized methods. Future work could adapt current techniques that involve nested sampling and make use of the combined-chain method.

CHAPTER 3

THERMODYNAMIC INTEGRATION-BASED METHODS

Thermodynamic integration, like nested sampling, is a numerical technique for evaluating model evidence integrals. The technique was originally developed by Kirkwood [1935] to estimate the free energy of a fluid. Various improvements and changes have been made over the decades, and the incarnation of the technique that is discussed in this chapter is described by Goggans and Chi [2004]. A derivation of the thermodynamic integral is given in Section 3.1, and the broad details of the [Goggans and Chi, 2004] method are discussed in Section 3.2. Goggans and Chi [2004]'s original implementation follows John Skilling's BayeSys [Skilling, 2004a], and both make use of binary slice sampling and the Hilbert curve to complete the implementation. This method, my parallel version of it, and my proposed alternative to the Hilbert curve are described in Section 3.4. Finally in Section 3.5, I propose a modification of the thermodynamic integration with adaptive annealing and importance sampling method that uses PyStan [Carpenter et al., 2017, Stan Development Team, 2018] and the No U Turn Sampler (NUTS) [Hoffman and Gelman, 2014] instead of binary slice sampling.

Portions of this chapter also appear in my conference paper *Using the Z-order curve for Bayesian model comparison* [Henderson and Goggans, 2018].

3.1 Thermodynamic Integration Derivation

In this section, I derive the thermodynamic integral form of the model evidence integral. From Bayes' theorem, for model vector $\boldsymbol{M}$, data vector $\boldsymbol{D}$, model parameter

vector $\boldsymbol{\Theta}$, and prior information I, we have

$$\mathrm{p}(\boldsymbol{D}|\boldsymbol{M}, I) = \int \mathrm{p}(\boldsymbol{D}|\boldsymbol{\Theta}, \boldsymbol{M}, I)\mathrm{p}(\boldsymbol{\Theta}|\boldsymbol{M}, I)\,\mathrm{d}\boldsymbol{\Theta}. \tag{3.1}$$

Now introduce an inverse temperature parameter, β, that will control how much the likelihood influences the evidence value,

$$\mathrm{p}(\boldsymbol{D}|\boldsymbol{M}, \beta, I) = \int \left[\mathrm{p}(\boldsymbol{D}|\boldsymbol{\Theta}, \boldsymbol{M}, I)\right]^{\beta} \mathrm{p}(\boldsymbol{\Theta}|\boldsymbol{M}, I)\,\mathrm{d}\boldsymbol{\Theta}. \tag{3.2}$$

If β is 0, the integrand of (3.2) is simply the prior. If β is 1, (3.2) is the standard evidence integral, as in (3.1).

Differentiate the log of the evidence with respect to β using the chain rule,

$$\frac{\mathrm{d}}{\mathrm{d}\beta} \log \mathrm{p}(\boldsymbol{D}|\boldsymbol{M}, \beta, I) = \frac{1}{\mathrm{p}(\boldsymbol{D}|\boldsymbol{M}, \beta, I)} \frac{\mathrm{d}}{\mathrm{d}\beta} \mathrm{p}(\boldsymbol{D}|\boldsymbol{M}, \beta, I). \tag{3.3}$$

Define a function we will call the energy,

$$E_L(\boldsymbol{\Theta}) = -\log \mathrm{p}(\boldsymbol{D}|\boldsymbol{\Theta}, \boldsymbol{M}, I). \tag{3.4}$$

Evaluate the derivative on the right-hand side of (3.3), substituting in the definition of the energy (3.4),

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\beta} \log \mathrm{p}(\boldsymbol{D}|\boldsymbol{M}, \beta, I) &= \frac{1}{\mathrm{p}(\boldsymbol{D}|\boldsymbol{M}, \beta, I)} \int \frac{\mathrm{d}}{\mathrm{d}\beta} \exp\left[-\beta E_L(\boldsymbol{\Theta})\right] \mathrm{p}(\boldsymbol{\Theta}|\boldsymbol{M}, I)\,\mathrm{d}\boldsymbol{\Theta}\,, && (3.5)\\
&= \int -E_L(\boldsymbol{\Theta}) \frac{\exp\left[-\beta E_L(\boldsymbol{\Theta})\right] \mathrm{p}(\boldsymbol{\Theta}|\boldsymbol{M}, I)}{\mathrm{p}(\boldsymbol{D}|\boldsymbol{M}, \beta, I)}\,\mathrm{d}\boldsymbol{\Theta}\,, && (3.6)\\
&= -\int E_L(\boldsymbol{\Theta}) \mathrm{p}(\boldsymbol{\Theta}|\boldsymbol{M}, \boldsymbol{D}, \beta, I)\,\mathrm{d}\boldsymbol{\Theta}\,. && (3.7)
\end{aligned}$$

(3.7) is the negative expectation of the energy with respect to the parameter posterior con-

ditioned on β,

$$\frac{\mathrm{d}}{\mathrm{d}\beta} \log \mathrm{p}(\boldsymbol{D}|\boldsymbol{M}, \beta, I) = -\left\langle E_L(\boldsymbol{\Theta}) \right\rangle_\beta \,. \tag{3.8}$$

To obtain the log-evidence, integrate (3.8) over the domain of β,

$$\log \mathrm{p}(\boldsymbol{D}|\boldsymbol{M}, \beta, I) = -\int_0^1 \left\langle E_L(\boldsymbol{\Theta}) \right\rangle_\beta \, \mathrm{d}\beta \,. \tag{3.9}$$

Equation (3.9) generally cannot be evaluated analytically, nor can the expected energy at each β within the integral be determined analytically. In practice, the following must be done to use (3.9) effectively:

- use MCMC to estimate the expected energy at each value of β,
- develop a fixed schedule for β or employ an adaptive strategy to choose optimal stepped values of β,
- and compute the quadrature estimate of the integral.

3.2 Adaptive Annealing and Importance Sampling

Much of the thermodynamic integration literature, e.g. [Gelman and Meng, 1998] and [Oates et al., 2016], describes methods that use fixed annealing schedules. That is, all of the values of β are decided before any sampling is done. For problems where the distributions involved are not overly complex, this fixed schedule TI works well. However, for problems in which the distributions are based on physical models, are multi-modal, have correlations in the parameters, or have pronounced curving degeneracies, using a fixed schedule can lead to significant error in the log-evidence estimate. Signal detection problems tend to result in complex distributions like this, so to accommodate these distributions, an adaptive temperature annealing schedule can be used. Goggans and Chi [2004] lay out a general procedure for deploying a thermodynamic integration method with adaptive annealing, which is described below.

1. Start at $\beta = 0$ where $\mathrm{p}(\boldsymbol{\Theta}|\boldsymbol{M}, \boldsymbol{D}, \beta, I) = \mathrm{p}(\boldsymbol{\Theta}|\boldsymbol{M}, I)$, and draw C samples from this distribution (the prior).

2. Compute the Monte Carlo estimator for the expected energy at the current β,

$$\langle E_L(\boldsymbol{\Theta}) \rangle_\beta \approx \frac{1}{N} \sum_{t=1}^{C} E_L(\boldsymbol{\Theta}_t)\,, \tag{3.10}$$

   where $\boldsymbol{\Theta}_t$ is the current position of the t-th Markov chain.

3. Increment β by $\Delta\beta_i$, where

$$\Delta\beta_i = \frac{\log \frac{\max w_j}{\min w_j}}{\max E_L(\boldsymbol{\Theta_i}) - \min E_L(\boldsymbol{\Theta_i})}, \tag{3.11}$$

   j is the index on the chains, w_j is the weight associated with chain j, and

$$w_j = \exp[-\Delta\beta_i E_L(\boldsymbol{\Theta_j})]. \tag{3.12}$$

4. Re-sample the population of samples using importance sampling.

5. Use MCMC to refresh the current population of samples. This yields a more accurate sampling of the distribution at the current temperature. This step can be easily parallelized, as each sample's position can be shifted independently of the others.

6. Return to step 2 and continue until β_i reaches 1.

7. Estimate (3.9) using quadrature and the expected energy estimates built up using (3.10).

3.2.1 Importance sampling with re-sampling

The importance sampling with re-sampling technique used in this method is described in [Goggans and Chi, 2004] and follows Liu et al. [2001]. Once a new value of β is chosen

according to the adaptive annealing procedure, importance sampling with re-sampling is used to sample the distribution under the new value of β. The importance weights used for re-sampling are

$$w_j = \frac{\mathrm{p}(\boldsymbol{\Theta}|\boldsymbol{M}, \boldsymbol{D}, \beta_{i+1}, I)}{\mathrm{p}(\boldsymbol{\Theta}|\boldsymbol{M}, \boldsymbol{D}, \beta_i, I)} = \exp(-\Delta\beta_i E_L(\boldsymbol{\Theta}_j)), \tag{3.13}$$

which are then normalized,

$$W_j = J\frac{w_j}{\sum_{j=1}^{J} w_j}. \tag{3.14}$$

The Monte Carlo approximation of the posterior distribution under the new β is then

$$\mathrm{p}(\boldsymbol{\Theta}|\boldsymbol{M}, \boldsymbol{D}, \beta_{i+1}, I) \approx \frac{1}{J}\sum_{j=1}^{J} W_j \delta(\boldsymbol{\Theta} - \boldsymbol{\Theta}_j). \tag{3.15}$$

At this point, the current population of samples needs to be re-sampled according to (3.15). In order to determine the number of each sample that should be kept, the following calculation is used,

$$N_j = \sum_{k=0}^{J-1}\left[\mathrm{U}\left(u + k - \sum_{i=1}^{j-1} W_i\right) - \mathrm{U}\left(u + k - \sum_{i=1}^{j} W_i\right)\right], \tag{3.16}$$

where u is a uniform random variate on $[0, 1]$, and U is the unit step function. If $N_j = 0$ for a particular sample, that sample is removed. If $N_j > 1$ for a particular sample, that sample is replicated. If $N = 1$ for a particular sample, that sample is simply kept with no change. A pseudo-code implementation of importance sampling with re-sampling is shown in Algorithm 3.1.

The importance sampling with re-sampling processes by nature can only replicate or remove existing samples. In order to accurately sample the posterior distribution, these samples need to be refreshed periodically, with their current positions serving as starting points. In the first method, a combination of binary slice sampling and leapfrog sampling accomplish this requirement. In the second method, the No U-Turn Sampler is used instead.

Algorithm 3.1 Importance sampling with re-sampling

```
1: function ImportanceSampling(w, α, E*, C)
2:     Sort w, α, and E* by w
3:     w ← (C/∑w)w
4:     u ← Rand(0, 1)
5:     w_old ← w
6:     for i ← 1, C do
7:         w_i ← ∑_k^i w_{old,i}
8:     end for
9:     j ← 0
10:    q ← −1
11:    for m ← 1, C do
12:        while w_m > u do
13:            α_j ← α_m
14:            E*_j ← E*_m
15:            q ← m
16:            u ← u + 1
17:            j ← j + 1
18:        end while
19:    end for
20: end function
```

3.2.2 Adaptive annealing

In step 3, a new value of β is chosen. The importance weights in (3.13) depend on the change in the value of β, and we would like the weights to be set such that, on average, 1 sample is discarded and replaced at each temperature. The β update equation (3.11) is designed with this goal in mind.

The ratio $W = \frac{\max w_j}{\min w_j}$ in (3.11) is a constant set by the user. This constant is the desired ratio of the maximum importance weight in the sample population to the minimum importance weight. As long as this ratio is slightly greater than 1 (e.g., 1.05), the importance sampling process will usually discard and replace no more than one sample per temperature, as is the goal. If the distribution being sampled is not particularly challenging, higher values for the ratio constant can be used for a significant decrease in run time.

The denominator in (3.11) allows the change in temperature to be controlled by the conditions encountered by the sampler. If the maximum energy and minimum energy are

close, it is likely that we are sampling the distribution effectively, and β can be changed by a larger amount without disrupting the overall sampling. However, if the values are far apart, the implication is that we are sampling a more difficult portion of the distribution, and that a more gradual change in β will better serve the overall sampling.

3.3 Representing the Model Parameters

Users of the techniques described in this chapter may wish to evaluate the probabilities for any of a wide variety of mathematical models. The parameters of these models can be assigned any (proper) prior distribution, as the user deems necessary. Ultimately, the techniques developed in this chapter need to be able to sample model parameter spaces according to these prior distributions. While this challenge can be met in several ways, we take an approach that involves introducing a parameter transformation step into the energy function calculation.

Within the TI-based methods proposed here, parameters are always represented as either integer values on $[0, 2^B]$ (for TI with Binary Slice Sampling) or floating point values on the interval $[0, 1]$ (for TI with Stan), each with an independent uniform prior distribution. A parameter transformation routine must be included in the energy calculation function that maps these internal parameter representations to values with the correct prior probabilities for computing the energy. If the desired true prior distribution is also uniform, the parameter transformation amounts to a simple scaling operation. Other prior distributions, such as Gaussian distributions, have similar straightforward mapping functions.

In the case that a specialized mapping function is not available, the prior cumulative distribution function (CDF) may be used in all cases to perform the mapping through a process known as inverse transform sampling. Let u be a variate drawn from Uniform$(0, 1)$, let $F_X(x)$ be the CDF for a prior distribution $f_X(x)$, and let $F_X^{-1}(x)$ be the functional inverse of the prior CDF. $F_X^{-1}(u)$ transforms the uniform variate into a variate drawn from $f_X(x)$.

3.4 Thermodynamic Integration with Binary Slice Sampling

Goggans and Chi [2004] do not specify how to carry out step 5, the refreshing of the population of samples, but their reference implementation takes inspiration from BayeSys by Skilling [2004a]. While BayeSys uses several different MCMC "engines" to carry out its sampling, the reference implementation of Goggans and Chi [2004] uses just two: binary slice sampling and leapfrog sampling. A pseudo-code listing of this procedure is shown in Algorithm 3.2.

Binary slice sampling [Skilling and MacKay, 2003] is an integer-based adaptation of slice sampling by Neal [2003]. Slice sampling provides a procedure for sampling distribution $f(x)$. Its procedure for moving from point $x_0 \in \mathcal{R}^n$ to another point $x_1 \in \mathcal{R}^n$ in a way that maintains detailed balance under distribution $f(x)$ is described below.

A new variable y is drawn uniformly from the interval $(0, f(x_0))$. The portions of the distribution $f(x)$ that are greater than this value y are considered part of the "slice" to be sampled. A stepping-out procedure if performed from x_0 to find points that are beyond the edges of the slice, then a stepping-in procedure is performed to find bounds that contain all or most of the slice. Once these bounds are obtained, the area defined is sampled uniformly to find the new point, x_1. This procedure is straightforward only in one dimension, though N-dimensional extensions do exist.

Skilling and MacKay [2003]'s binary slice sampling follows this procedure, but it uses integer values for x and bit operations to perform the stepping maneuvers and random sampling. A pseudo-code implementation of binary slice sampling is shown in Algorithm 3.3. This implementation uses a space-filling curve to map multi-dimensional coordinates to a single large integer value, allowing binary slice sampling to be used to sample multi-dimensional distributions without any modification.

It should be noted that the reference TI implementation by Goggans and Chi [2004], BayeSys by Skilling [2004a], and my TI-based methods all omit the stepping-out portion of slice sampling. Instead of first stepping out, my TI-based methods begin by setting the slice

Algorithm 3.2 Thermodynamic integration with binary slice sampling

procedure TI($P, M, S, N, C, B, W, data$)
  Inputs: P–Number of parameters, M–Number of chains steps, S–Number of slice sampling steps, N–New origin probability, C–Number of chains, B–Number of bits per parameter, W–Ratio to control adaptive annealing, $data$–Data
  for $m \leftarrow 1, C$ **do**
    $X \leftarrow \textsc{RandInt}(0, 2^{PB} - 1)$
    $\alpha^m \leftarrow \textsc{LineToAxes}(X, B, P)$
    $E^*_m \leftarrow \textsc{Energy}(\alpha^m, data)$
  end for
  $i \leftarrow 1$
  Compute $\langle E^* \rangle_i$
  $\beta_1 \leftarrow \min\{\log(W)/[\max(E^*) - \min(E^*)], 1\}$
  $w \leftarrow \exp(-\beta_1 E^*)$
  $\textsc{ImportanceSampling}(w, \alpha, E^*, C)$
  while $\beta_i > 0$ and $\beta_i < 1$ **do**
    for $i \leftarrow 1, M$ **do**
      for $m \leftarrow 1, C$ **do**
        $\textsc{BinarySliceSampling}(\alpha^m, E^*_m, B, C, P, S, N, \beta_i, data)$
      end for
      $\textsc{LeapFrog}(\alpha, E^*, B, C, P, data)$
    end for
    $i \leftarrow i + 1$
    $\Delta\beta \leftarrow \log(W)/[\max(E^*) - \min(E^*)]$
    $\beta_i \leftarrow \min(\beta_{i-1} + \Delta\beta, 1)$
    if $\beta_{i-1} + \Delta\beta > 1$ **then**
      $\Delta\beta \leftarrow 1 - \beta_{i-1}$
    end if
    $w \leftarrow \exp(-\Delta\beta E^*)$
    $\textsc{ImportanceSampling}(w, \alpha, E^*, C)$
  end while
  Estimate (3.9) using trapezoid rule and $\{\beta_i\}$ and $\{\langle E^* \rangle_i\}$
end procedure

Algorithm 3.3 Binary slice sampling

function BINARYSLICESAMPLING($\alpha, E^*, B_{in}, C, P, S, N, \beta, data$)
  $B \leftarrow 2^{PB_{in}}$
  $X_{orig} \leftarrow \text{RANDINT}(0, B)$
  $\alpha_{orig} \leftarrow \text{LINETOAXES}(X_{orig}, B_{in}, P)$
  $\alpha_i \leftarrow \alpha$
  for $i \leftarrow 1, S$ **do**
    if $\text{RAND}(0,1) < N$ **then**
      $X_{orig} \leftarrow \text{RANDINT}(0, B)$
      $\alpha_{orig} \leftarrow \text{LINETOAXES}(X_{orig}, B_{in}, P)$
    end if
    $\alpha_i \leftarrow (\alpha_i - \alpha_{orig}) \mod 2^{B_{in}}$
    $X \leftarrow \text{AXESTOLINE}(\alpha_i, B_{in}, P)$
    $U \leftarrow \text{RANDINT}(0, B)$
    $y \leftarrow \beta\text{ENERGY}(\alpha, data) - \log(\text{RAND}(0,1))$
    $l \leftarrow PB_{in}$
    $E^{*\prime} \leftarrow \infty$
    while $\beta E^{*\prime} > y$ and $l > 1$ **do**
      $N \leftarrow \text{RANDINT}(0, 2^l)$
      $X' \leftarrow (\{[(X - U) \mod B] \oplus N\} + U) \mod B$
      $\alpha_i \leftarrow \text{LINETOAXES}(X', B_{in}, P)$
      $\alpha_i \leftarrow (\alpha_i + \alpha_{orig}) \mod 2^{B_{in}}$
      $E^{*\prime} \leftarrow \text{ENERGY}(\alpha_i, data)$
      $l \leftarrow l - P$
    end while
    $\alpha \leftarrow \alpha_i$
  end for
end function

width as the maximum range allowed by the prior and step in from there.

This TI implementation also uses leapfrog sampling to help shuffle the sample population more effectively. Leapfrog sampling uses samples' neighbors to generate new trial positions, and it works as follows. The population is sorted according to each sample's line (Hilbert or Z-order) index. For each sample in the sorted list, the left and right neighbors are determined. The midpoint between the two neighbor samples in real parameter space is found, and the sample is reflected across that midpoint. If the new position is still between the left and right neighbors, a Metropolis acceptance test is carried out, and the new position is either accepted or rejected. A pseudo-code implementation of the leapfrog sampling method is shown in Algorithm 3.4. Algorithm 3.2 allows for several Binary slice sampling (BSS) steps per leapfrog step to allow for good mixing.

3.4.1 Space-filling curves

A space-filling curve is a continuous and nowhere-differentiable function that maps the unit line to a unit hypercube of arbitrary dimension. When I refer to space-filling curves from here on in this chapter, I am referring to something related but slightly different: an approximation to a true space-filling curve that takes the form $f : \mathbb{N}_0^1 \to \mathbb{N}_0^N$ and maps the 1-dimensional natural numbers to the N-dimensional natural numbers. This approximation to the space-filling curve allows multidimensional probability distributions to be sampled using one-dimensional sampling techniques, such as binary slice sampling. Specifically, it allows us to evaluate model probabilities for models with multiple parameters by creating a 1-to-1 mapping from the multidimensional parameter space to a one-dimensional integer index. The ideal space-filling curve for this application would have the following properties:

- Locality. Points that are nearby in $\mathbb{N}_0^N$ should be nearby on the curve index in $\mathbb{N}_0^1$ as well. The converse should be true as well.

- Time-efficiency. The algorithms for performing the mapping between parameter space and curve indexes should be time-efficient.

Algorithm 3.4 Leapfrog sampling function

```
1: procedure LEAPFROG(α, E*, B, C, P, data)
2:     for m ← 1, C do
3:         X_m ← AXESTOLINE(α_m, B, P)
4:     end for
5:     Sort X
6:     for m ← 1, C do
7:         α_m ← LINETOAXES(X_m, B, P)
8:         E*_m ← ENERGY(α_m, data)
9:     end for
10:    for m ← 1, C do
11:        αcur ← α_m
12:        if m = 1 then
13:            l ← α_C
14:        else
15:            l ← α_{m−1}
16:        end if
17:        if m = C then
18:            r ← α_1
19:        else
20:            r ← α_{m+1}
21:        end if
22:        Xl ← AXESTOLINE(l, B, P)
23:        Xr ← AXESTOLINE(r, B, P)
24:        αnew ← (l + r − αcur) mod 2^B
25:        Xnew ← AXESTOLINE(αnew, B, P)
26:        if (m = 1 and (Xnew > Xl or Xnew < Xr)) or (m = C and (Xnew > Xl or
       Xnew < Xr)) or (m > 1 and m < C and Xnew > Xl and Xnew < Xr then
27:            Enew ← ENERGY(αnew, data)
28:            if Enew < E*_m then
29:                E*_m ← Enew
30:                α_m ← αnew
31:            else
32:                u ← RAND(0, 1)
33:                if Enew − E*_m < − log(u) then
34:                    E*_m ← Enew
35:                    α_m ← αnew
36:                end if
37:            end if
38:        end if
39:    end for
40: end procedure
```

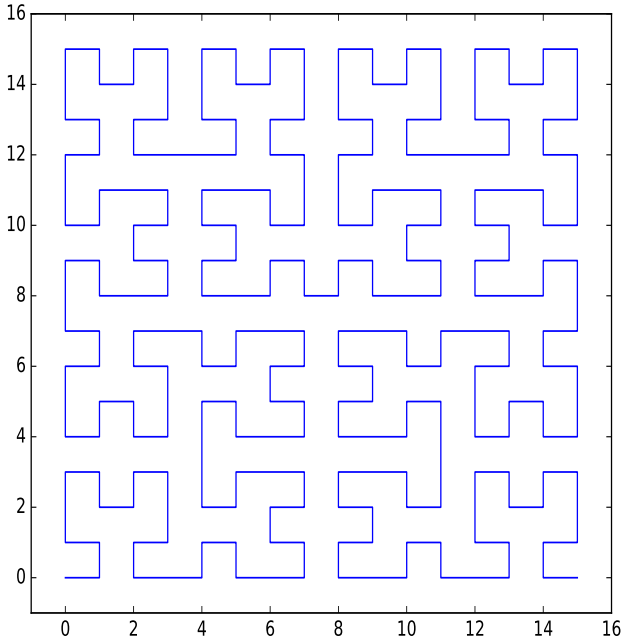


Figure 3.1: Hilbert curve for 2 dimensions with 4 bits per dimension.

- Bi-directionality. Algorithms should exist for mapping parameter space to curve indexes and from curve indexes to parameter space.

3.4.1.1 Hilbert Curve

The discrete approximation of the Hilbert curve (hereafter referred to simply as the Hilbert curve) [Sagan, 1994, chapter 2], [Skilling, 2004c] is a space-filling curve that has good locality properties. If two indexes are consecutive on the Hilbert curve, the points in parameter space that correspond to them are adjacent. There are also bi-directional transform functions available for the Hilbert curve, and these transform functions can be implemented in a time-efficient way. An example 4 bit per dimension Hilbert curve for a 2-dimensional parameter space is shown in Figure 3.1. A pseudo-code implementation of the Hilbert curve index-to-parameter transformation is shown in Algorithm 3.5.

Algorithm 3.5 Hilbert curve line-to-axes function

```
1: function LineToAxes(Line, B, P)
2:     for i ← 1, P do
3:         linen_{P−i} ← (Line ≫ (B(i − 1)))  mod 2^B
4:     end for
5:     M ← 1 ≪ B − 1
6:     for i ← 1, P do
7:         X_i ← 0
8:     end for
9:     q ← 0,    p ← M
10:    for i ← 1, P do
11:        j ← M
12:        while j > 0 do
13:            if linen_i ∧ j then
14:                X.q ← X.q ∨ p
15:            end if
16:            q ← q + 1
17:            if q = n then
18:                q ← 0,    p ← p ≫ 1
19:            end if
20:            j ← j ≫ 1
21:        end while
22:    end for
23:    t ← X_P ≫ 1
24:    for i ← P, 2 do
25:        X_i ← X_i ⊕ X_{i−1}
26:    end for
27:    X_1 ← X_1 ⊕ t,    M ← 2 ≪ (B − 1),    Q ← 2
28:    while Q ≠ M do
29:        P ← Q − 1
30:        for i ← P, 2 do
31:            if X_i ∧ Q then
32:                X_1 ← X_1 ⊕ P
33:            else
34:                t ← (X_1 ⊕ X_i) ∧ P,    X_1 ← X_1 ⊕ t,    X_i ← X_i ⊕ t
35:            end if
36:        end for
37:        if X_1 ∧ Q then
38:            X_1 ← X_1 ⊕ P
39:        end if
40:        Q ← Q ≪ 1
41:    end while
42:    return X
43: end function
```

```
Z-order index       Axes coordinates

                    adgj
abcdefghijkl  <-->  behk
                    cfil
```

Figure 3.2: Example of Z-order bit-interleaving for 3 dimensions with 4 bits per dimension

3.4.1.2 Z-order Curve

The Z-order curve [Sagan, 1994, chapter 5] (also known as the Lebesgue curve or Morton curve) is a space-filling curve that maintains locality somewhat less well than the Hilbert curve but meets our other two requirements well. Most importantly, its transformation algorithms are faster than those for the Hilbert curve. In order to transform from the N-dimensional parameter space to the 1-dimensional Z-order curve, the bits of the integer coordinates for each dimension are interleaved. If the parameter space has 3 dimensions and each coordinate axis is represented by a 4-bit integer, the resulting Z-order curve representation will be a 12-bit integer. An example of this bit-interleaving is shown in Figure 3.2, in which each letter represents a binary digit, demonstrates the bit interleaving described above. An example of a Z-order curve for 2 dimensions with 4 bits per dimension is shown in Figure 3.3.

The simple way to perform the Z-order mapping is to loop over the bit and axis indexes and place each bit where it needs to be individually. However, this method does not provide a time complexity improvement over the Hilbert curve transform functions. A cleverer, bitmask-based approach exists, and it is documented in several places online. The most thorough description of an algorithm for generating the necessary bitmasks for arbitrary numbers of dimensions and bits per dimension is given in a Stackoverflow answer by user Gabriel [2013]. Gabriel also describes the general method by which the mapping is performed using the bitmasks, but he does not provide algorithms for doing so. The following list outlines the basic procedure for mapping from the Z-order index to axes coordinates:

1. Generate bitmasks based on number of bits b and number of parameters n.

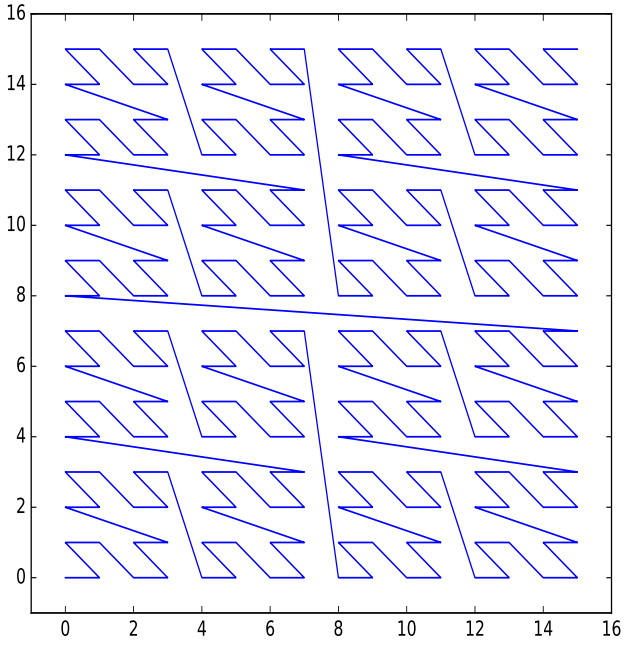


Figure 3.3: Z-order curve for 2 dimensions with 4 bits per dimension.

2. AND the first mask with the Z-order integer to select only every n bits.

3. Loop over each mask. For the ith mask, XOR the Z-order integer with itself shifted to the right by i, then mask the result.

4. Shift the original Z-order integer to the right by 1, then repeat the above from step 2 for each dimension.

A pseudo-code implementation for the bitmask computation function is shown in Algorithm 3.6, and a pseudo-code implementation for the bitmask-based line-to-axes transformation function is shown in Algorithm 3.7.

3.4.2 Parallel implementation

In implementations of this binary slice sampling and space-filling curve-based technique, most of the computation time is taken up by the line-to-axes and axes-to-line operations. Because the vast majority of the invocations of these operations occur during the

Algorithm 3.6 Z-order curve mask computation function

```
1: function ComputeBitMask(B, P)
2:     P ← P − 1
3:     for i ← 1, B do
4:         bd_i ← (i + 1)P    ▷ Stored as binary strings, with the leftmost two bits discarded
5:     end for
6:     maxLength ← length of longest string in bd
7:     moveBits ← empty list
8:     for i ← 1, maxLength do
9:         Append an empty list to moveBits
10:        for j ← 1, P do
11:            if Length(bd_j) ≥ i then
12:                if The ith bit from the end of bd_j is 1 then
13:                    Append j to moveBits_i
14:                end if
15:            end if
16:        end for
17:    end for
18:    for i ← 1, B do
19:        bitPositions_i ← i
20:    end for
21:    maskOld ← (1 ≪ B) − 1
22:    bitmasks ← empty list
23:    for i ← Length(moveBits), 1 do
24:        if Length(moveBits_i) > 0 then
25:            shifted ← 0
26:            for bitIdxToMove ∈ moveBits_i do
27:                shifted ← shifted ∨ (1 ≪ bitPositions_bitIdxToMove)
28:                bitPositions_bitIdxToMove ← bitPositions_bitIdxToMove + 2^i
29:            end for
30:            nonshifted ← ¬shifted ∧ maskOld
31:            shifted ← shifted ≪ 2^i
32:            maskNew ← shifted ∨ nonshifted
33:            Append maskNew to bitmasks
34:            maskOld ← maskNew
35:        end if
36:    end for
37:    return bitmasks
38: end function
```

Algorithm 3.7 Z-Order curve line-to-axes function

```
 1: function LineToAxes(z, B, P)
 2:     if P = 1 then
 3:         return z
 4:     end if
 5:     masks ← ComputeBitMask(B, P)          ▷ Call only once for each B and P
 6:     Pop final entry from masks list into first
 7:     Reverse the masks list
 8:     Append 1 ≪ B to masks
 9:     minshift ← P − 1
10:     for i ← 1, P do
11:         zz ← z ≫ i
12:         zz ← zz ∧ first
13:         shift ← minshift
14:         for mask ∈ masks do
15:             zz ← (zz ⊕ (zz ≫ shift)) ∧ mask
16:             shift ← shift ≪ 1
17:         end for
18:         α_i ← zz
19:     end for
20:     return α
21: end function
```

binary slice sampling routine itself, the for loop on lines 15, 16, and 17 in Algorithm 3.2 is a natural place to parallelize the algorithm. Parallelization at this point allows for much larger sample populations (C), which improves the accuracy of the evidence estimate. The Thermodynamic integration with binary slice sampling (TI-BSS) example results in Section 3.6 are produced with a parallel implementation in this vein. Section 4.2 explores how much parallel speed-up can be gained for a given value of C.

3.5 Thermodynamic Integration with Stan

BayeSys, which was the inspiration for much of the previously described implementation of thermodynamic integration with binary slice sampling, was released almost two decades ago, and since then, new Markov chain Monte Carlo techniques and new implementations of existing techniques have been developed. In Summer 2018, I wanted to find out what statisticians considered to be the gold standard for MCMC software. After a survey of

the available tools, it seemed to be widely agreed that MC Stan (or simply Stan), developed by Carpenter et al. [2017], was the answer.

Stan uses the NUTS [Hoffman and Gelman, 2014] as the basis for its sampling functions. NUTS is based on Hamiltonian Monte Carlo (HMC) [Neal, 2011], which uses the gradient of the log-likelihood function to more efficiently explore the posterior distribution. NUTS improves upon HMC by automatically choosing optimal values for HMC's tunable method parameters. NUTS has been shown to sample complex distributions effectively. I sought to build an improved thermodynamic integration implementation by using Stan instead of binary slice sampling and leapfrog sampling to refresh the sample population at each temperature within TI. The result, TI-Stan, is described in this section.

The TI-Stan algorithm is shown in 3.8. My implementation is in Python, so I made use of the PyStan interface to Stan, developed by the Stan Development Team [2018]. Stan defines its own language for defining statistical models, which allows it to efficiently compute the derivatives needed for HMC via automatic differentiation. For a particular problem, it is therefore necessary to write a Stan file that contains the Stan-formatted specification of the model, in addition to the pure-Python energy functions necessary for TI-BSS. Once one is familiar with the simple Stan language, this additional programming cost becomes trivial compared to the time savings achieved by using this method instead of TI-BSS.

3.6 Examples

To test the accuracy of these TI-based methods, the same set of examples used in Chapter 2 are used here: the egg-crate distribution ((2.52), Figure 2.2a), the twin-Gaussian shell distribution ((2.58), Figure 2.8), and the multiple stationary frequency signal detection problem ((2.57), Figure 2.3).

For all of these examples, the settings used for TI-BSS are shown in Table 3.1, while the settings used for TI-Stan are shown in Table 3.2. For each example, the user-defined constant W was set to both 1.5 and 2.0. Box-plots are used extensively in this section. In

Algorithm 3.8 Thermodynamic integration with Stan

procedure TI($P, S, C, W, data$)
  Inputs: P–Number of parameters, S–Number of Stan iterations per temperature, C–Number of chains, W–Ratio to control adaptive annealing, *data*–Data
  for $m \leftarrow 1, C$ **do**
    $X \leftarrow \text{RandInt}(0, 2^{PB} - 1)$
    $\alpha^m \leftarrow \text{LineToAxes}(X, B, P)$
    $E^*_m \leftarrow \text{Energy}(\alpha^m, data)$
  end for
  $i \leftarrow 1$
  Compute $\langle E^* \rangle_i$
  $\beta_1 \leftarrow \min\{\log(W)/[\max(E^*) - \min(E^*)], 1\}$
  $w \leftarrow \exp(-\beta_1 E^*)$
  ImportanceSampling(w, α, E^*, C)
  while $\beta_i > 0$ and $\beta_i < 1$ **do**
    for $m \leftarrow 1, C$ **do**
      StanSampling($\alpha^m, E^*_m, C, P, S, \beta_i, data$)
    end for
    $i \leftarrow i + 1$
    $\Delta\beta \leftarrow \log(W)/[\max(E^*) - \min(E^*)]$
    $\beta_i \leftarrow \min(\beta_{i-1} + \Delta\beta, 1)$
    if $\beta_{i-1} + \Delta\beta > 1$ **then**
      $\Delta\beta \leftarrow 1 - \beta_{i-1}$
    end if
    $w \leftarrow \exp(-\Delta\beta E^*)$
    ImportanceSampling(w, α, E^*, C)
  end while
  Estimate (3.9) using trapezoid rule and $\{\beta_i\}$ and $\{\langle E^* \rangle_i\}$
end procedure

Table 3.1: Parameters for TI-BSS examples

Parameter	Value	Definition
S	200	Number of binary slice sampling steps
M	2	Number of combined binary slice sampling and leapfrog steps
C	256	Number of chains
B	32	Number of bits per parameter in SFC

Table 3.2: Parameters for TI-Stan examples

Parameter	Value	Definition
S	200	Number of steps allowed in Stan
C	256	Number of chains

these box-plots, the middle line represents the median value, the box is bounded by the upper and lower quartiles, and the whiskers extend to the range of the data that lies within 1.5 times the inter-quartile range. Any data points past this threshold are plotted as circles.

These results were generated on a Google Cloud instance with 32 virtual Intel Broadwell CPUs and 28.8 GB of RAM.

3.6.1 Eggcrate Likelihood

The true value of the log-evidence for this log-likelihood function is known up to five significant digits:

$$\log Z_{\text{true}} = 235.88. \tag{3.17}$$

A box-plot summarizing the log-evidence estimates over 20 runs each for the TI-Stan, TI-BSS-H, and TI-BSS-Z methods and for each value of W is shown in Figure 3.4. Figure 3.4 demonstrates that all of the TI methods at these settings underestimate the log-evidence, with TI-BSS-Z with $W = 1.5$ coming the closest. Likely a value of W closer to 1 would yield better results. It is also apparent that the precision in these results do not correlate with the accuracy, suggesting that for problems with unknown answers, high precision over multiple

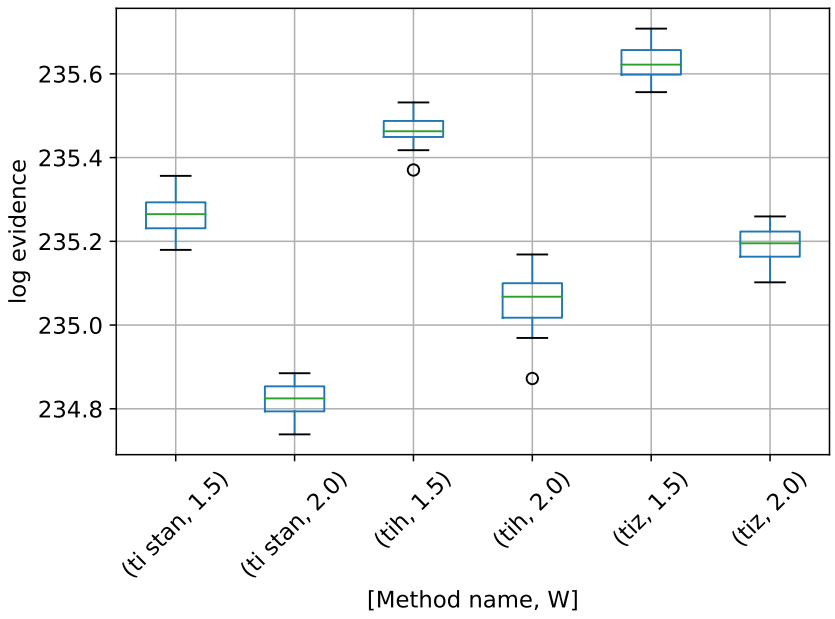


Figure 3.4: Box-plot of log-evidence for the egg-crate problem for each TI method

runs should not be interpreted as a proxy for accuracy.

A box-plot summarizing the run times over 20 runs each for the TI methods is shown in Figure 3.5. Figure 3.5 shows that each method speeds up considerably with a higher value of W. TI-BSS-H took the most time here, with TI-Stan and TI-BSS-Z running much more quickly.

3.6.2 Twin Gaussian Shells

This problem is run in 10 dimensions, 30 dimensions, and 100 dimensions. For the case of 10 dimensions, results are presented for all TI-BSS-H and TI-Stan, but not for TI-BSS-Z. For both values of W, TI-BSS-Z ended early in all its runs, and its estimate of the log-evidence unusably inaccurate. For the cases of 30 and 100 dimensions, results are presented only for TI-Stan, because a run of TI-BSS-H took too much time for it to be feasible to complete multiple runs.

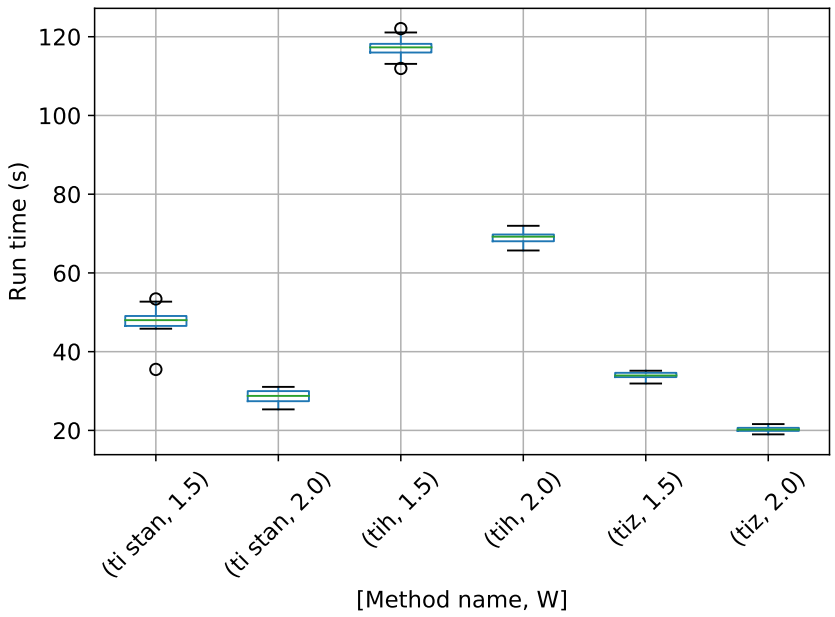


Figure 3.5: Box-plot of run time in seconds for the egg-crate problem for each TI method

3.6.2.1 10-dimensional twin Gaussian shells

We start with the 10-dimensional variant. A box-plot summarizing the log-evidence estimates over 20 runs each for TI-Stan and TI-BSS-H and for each value of W is shown in Figure 3.6. From [Feroz et al., 2009], the analytical log-evidence for this distribution is -14.59. None of the configurations tested actually reached that value, but the runs using $W = 1.5$ got closest, suggesting that a value of W closer to 1 would perhaps approach the correct value.

A box-plot summarizing the run times over 20 runs each for the TI methods is shown in Figure 3.7. Figure 3.7 shows the same trend with W as in the egg-crate problem, i.e., that the run time drastically increases as W approaches 1. It also shows that TI-BSS-H takes about 6 times longer, on average, than TI-Stan to compute its estimate of the log-evidence. According to Figure 3.6, the two methods have comparable accuracy and precision, so this difference in run time illustrates the difficulty the Hilbert curve-based method has with distributions of high dimension.

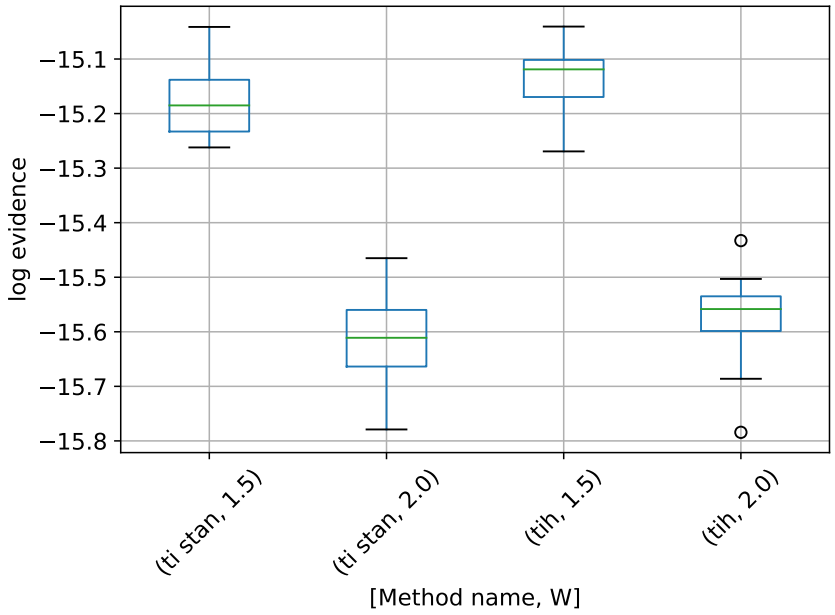


Figure 3.6: Box-plot of log-evidence for the 10-D twin Gaussian shell problem for TI-Stan and TI-BSS-H

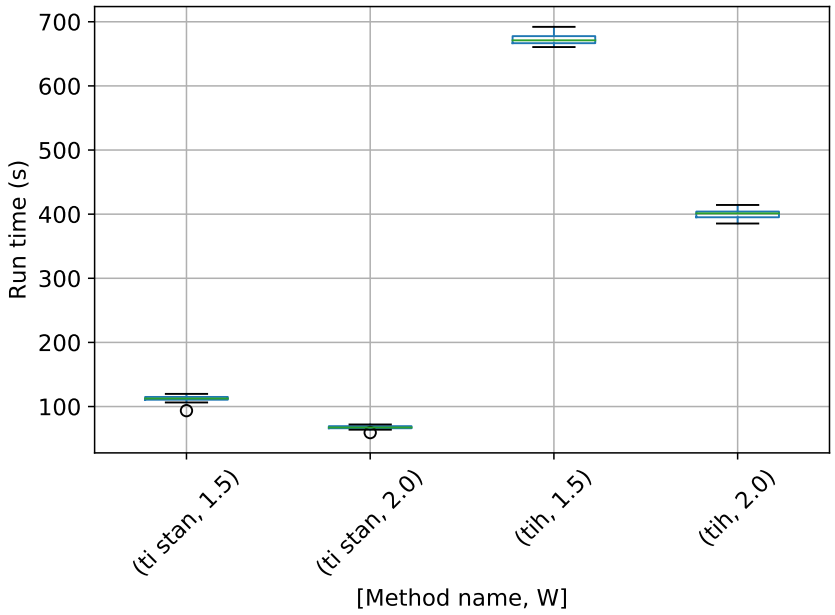


Figure 3.7: Box-plot of run time in seconds for the 10-D twin Gaussian shell problem for TI-Stan and TI-BSS-H

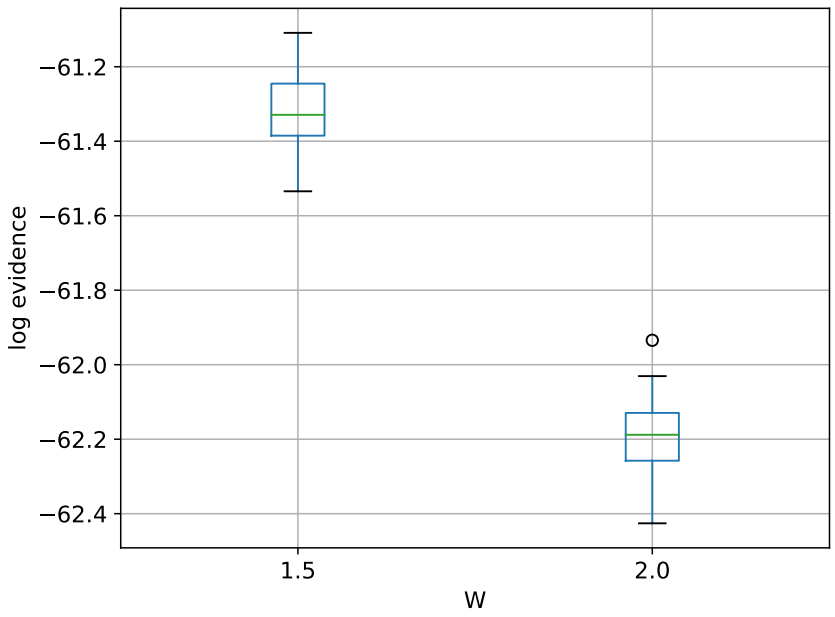


Figure 3.8: Box-plot of log-evidence for the 30-D twin Gaussian shell problem for TI-Stan

3.6.2.2 30-dimensional twin Gaussian shells

Regarding the 30-dimensional variant, the limitations of TI-BSS become even more apparent, as one run could not finish in a reasonable amount of time. TI-Stan, however, has no problem with the 30-D variant. A box-plot summarizing the log-evidence estimates over 20 runs each for TI-Stan at two values of W is shown in Figure 3.8. Again from [Feroz et al., 2009], the analytical log-evidence value is -60.13. Neither value of W allows TI-Stan to come within 1 unit of the correct answer here, and the gap is actually slightly larger here than in the 10-D variant. Again, a smaller value of W would probably be helpful here.

A box-plot summarizing the run times over 20 runs each for TI-Stan is shown in Figure 3.9. The common trend in run time vs W is observed here as well. TI-Stan takes about 3 times as long on average to compute log-evidence values in the 30-D case compared to the 10-D case, suggesting a possible linear relationship between run time and the number of dimensions for this problem.

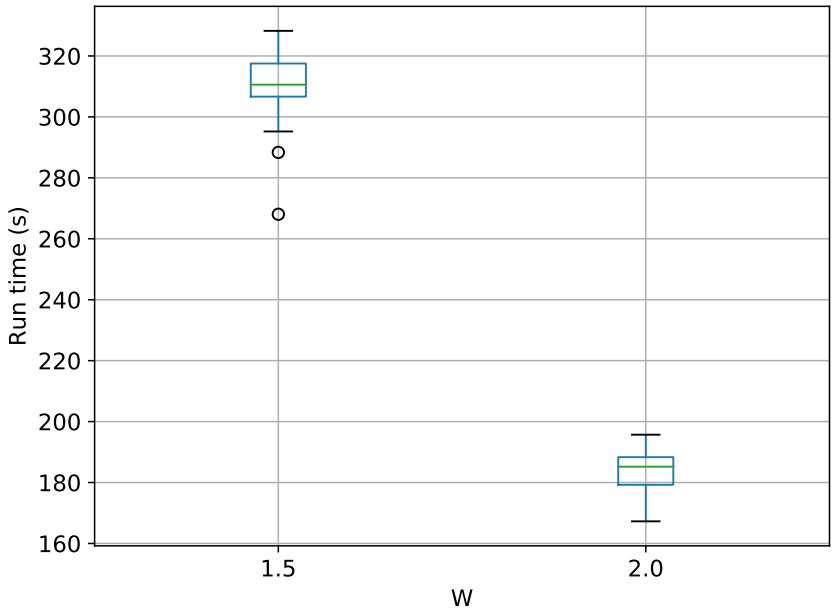


Figure 3.9: Box-plot of run time in seconds for the 30-D twin Gaussian shell problem for TI-Stan

3.6.2.3 100-dimensional twin Gaussian shells

A box-plot summarizing the log-evidence estimates over 20 runs each for TI-Stan at two values of W is shown in Figure 3.10. There is no analytical value available in the literature for this variant of the problem, but the result here seems believable. If it follows the trend of the previous examples, the log-evidence has been underestimated by some margin.

A box-plot summarizing the run times over 20 runs each for TI-Stan is shown in Figure 3.11. The results in Figure 3.11 disprove the hypothesis that the run time is related linearly to the number of parameters in this problem. The established relationship between run time and W continues here. This example is only a toy problem, but these results suggest that TI-Stan could be useful in problems with actual data and very high-dimensional models.

3.6.3 Detection of Multiple Stationary Frequencies

The final example problem is the multiple stationary frequencies detection problem described in subsection 2.3.2. Box-plots of log-evidence values for a model assuming one, two,

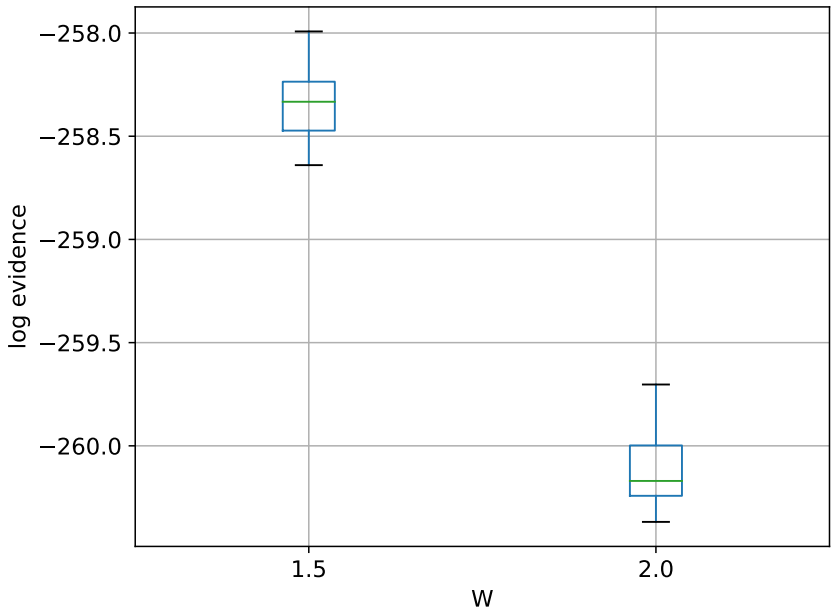


Figure 3.10: Box-plot of log-evidence for the 100-D twin Gaussian shell problem for TI-Stan

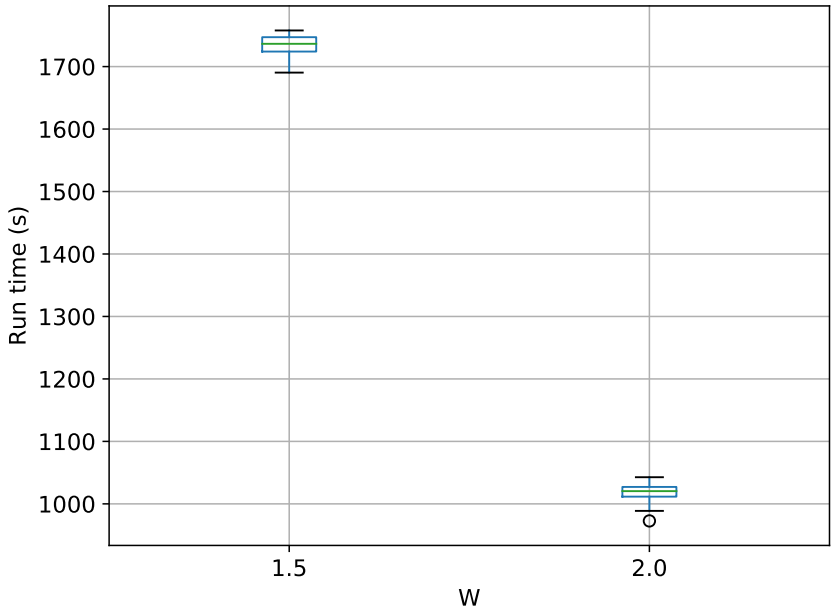


Figure 3.11: Box-plot of run time in seconds for the 100-D twin Gaussian shell problem for TI-Stan

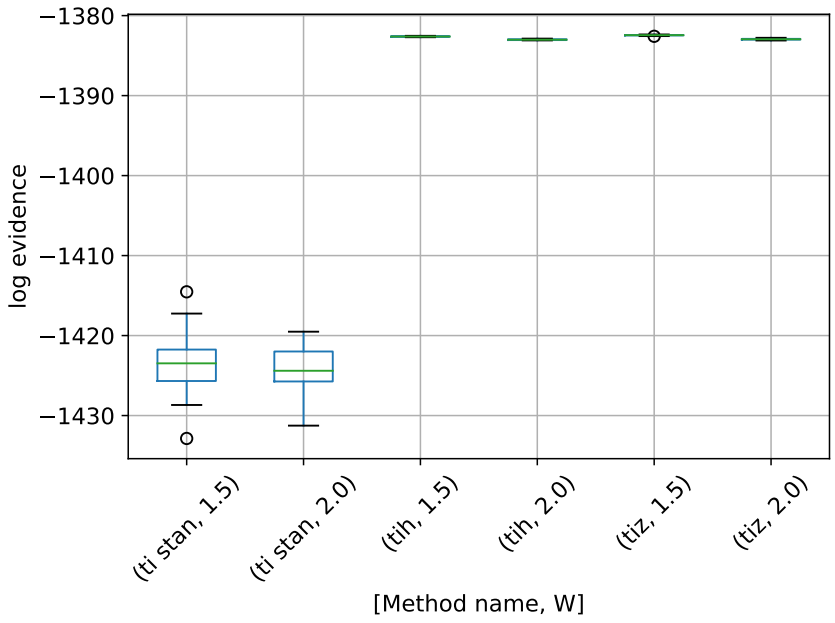


Figure 3.12: Box-plot of log-evidence for the one stationary frequency model for TI-Stan, TI-BSS-H, and TI-BSS-Z, for two values of W

and three frequencies present are shown in Figures 3.12, 3.13, and 3.14. For the models with one and three frequencies present, results are shown for TI-Stan, TI-BSS-H, and TI-BSS-Z. For the model with two frequencies present (the model also used to generate the test signal), results for TI-BSS-Z are not shown. As in the Gaussian shell problems, TI-BSS-Z ended early here and did not arrive at a reasonable result.

There are no analytical log-evidence values available for this example. I argue that a method is successful if the model used to generate the data clearly has the highest log-evidence, with a good margin between it and the log-evidence for the other models. There is some significant disagreement among the various methods for the "wrong" models (those with one and three frequencies), but the methods are in much closer agreement for the two frequency model. For TI-Stan and TI-BSS-H and for both values of W, the two frequency model is clearly the maximum-log-evidence choice. Even with the variations in the runs, the results do not overlap at any point from model to model, and the closest model-to-model margins are all greater than 2.3, which corresponds to an odds of 10.

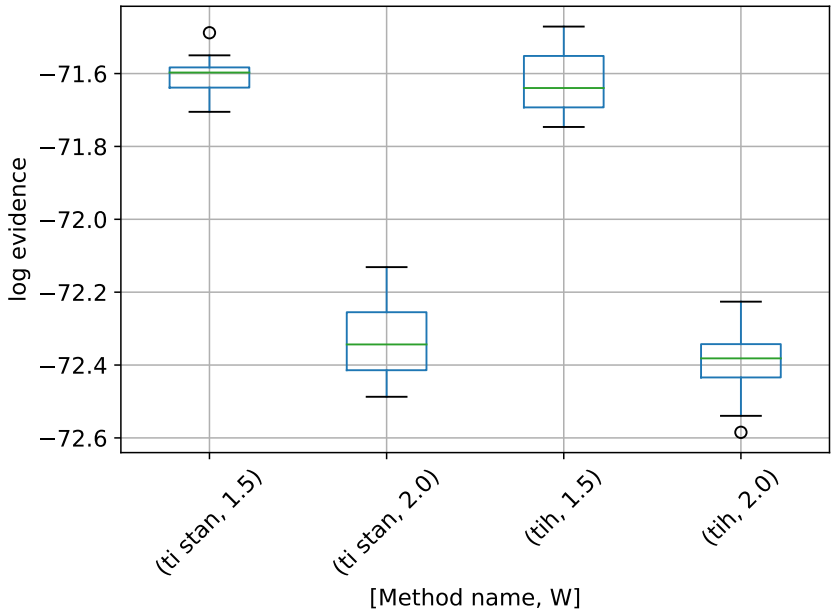


Figure 3.13: Box-plot of log-evidence for the two stationary frequency model for TI-Stan and TI-BSS-H, for two values of W

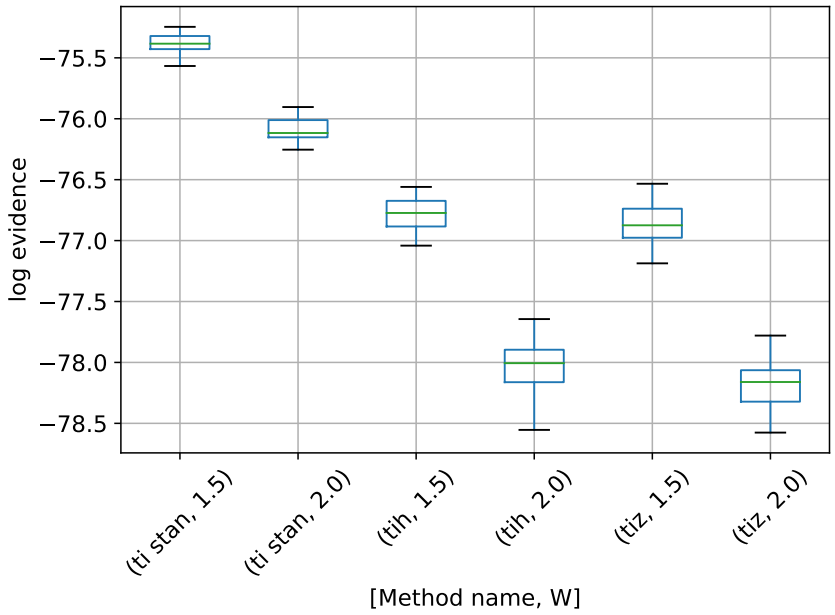


Figure 3.14: Box-plot of log-evidence for the three stationary frequency model for TI-Stan, TI-BSS-H, and TI-BSS-Z, for two values of W

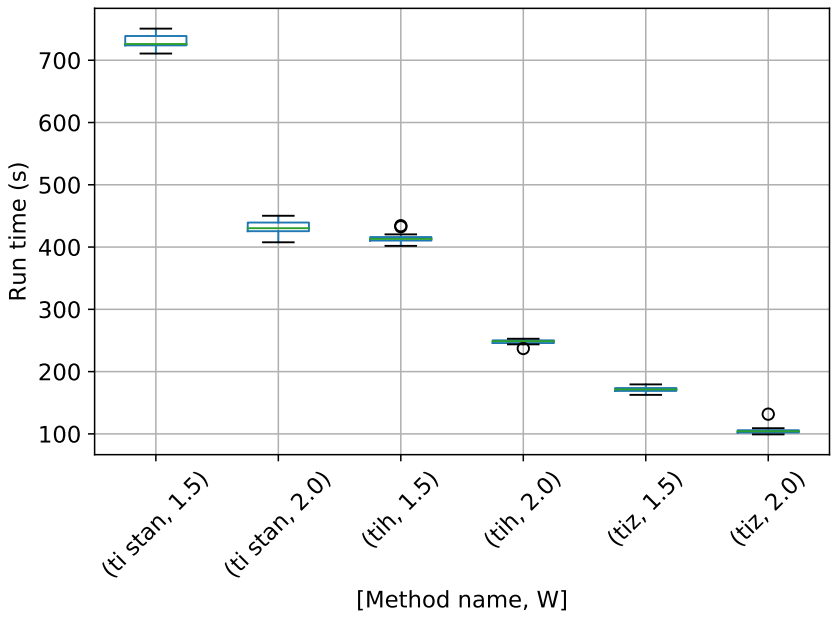


Figure 3.15: Box-plot of run time for the one stationary frequency model for TI-Stan, TI-BSS-H, and TI-BSS-Z, for two values of W

Box-plots of the run time for models assuming one, two, and three frequencies present are shown in Figures 3.15, 3.16, and 3.17. In Figure 3.15, TI-Stan has the greatest run time for both values of W, suggesting that its adaptive sampling process had trouble efficiently sampling distributions based on this high-error model. TI-BSS-H was much faster, and TI-BSS-Z was faster still. In Figure 3.16, the run times of TI-Stan and TI-BSS-H are comparable. This suggests that TI-Stan was able to more effectively sample the distribution based on the lower-error model. Figure 3.17 shows a similar pattern in the run times to Figure 3.15. The fact that this model is able to fit the noise in the data (yielding especially sharp distributions) and the fact that the distribution is increasingly multi-modal as the number of frequencies increases may explain why TI-Stan took a long time to compute a result here.

3.7 Conclusion

This chapter has presented three distinct model comparison algorithms based on thermodynamic integration: TI-BSS-H, TI-BSS-Z, and TI-Stan. Among these, TI-Stan is the

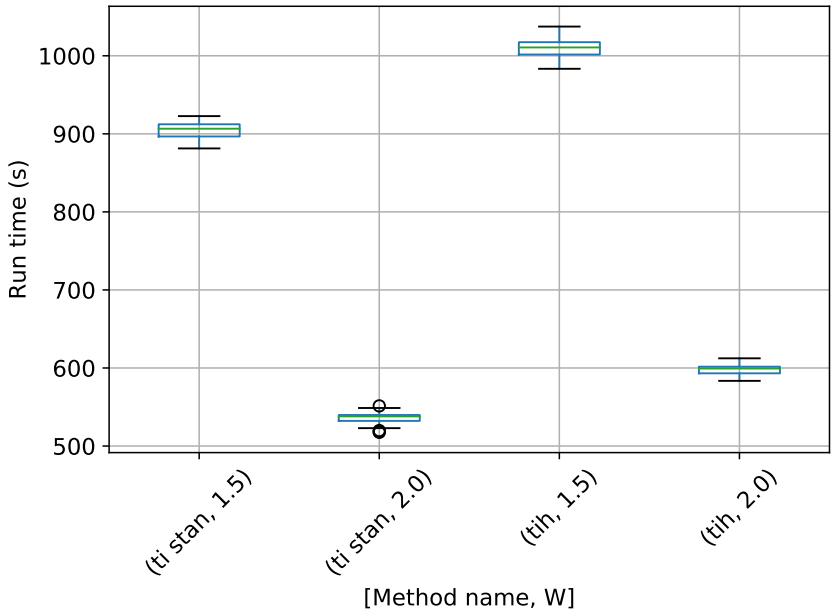


Figure 3.16: Box-plot of run time for the two stationary frequency model for TI-Stan and TI-BSS-H, for two values of W

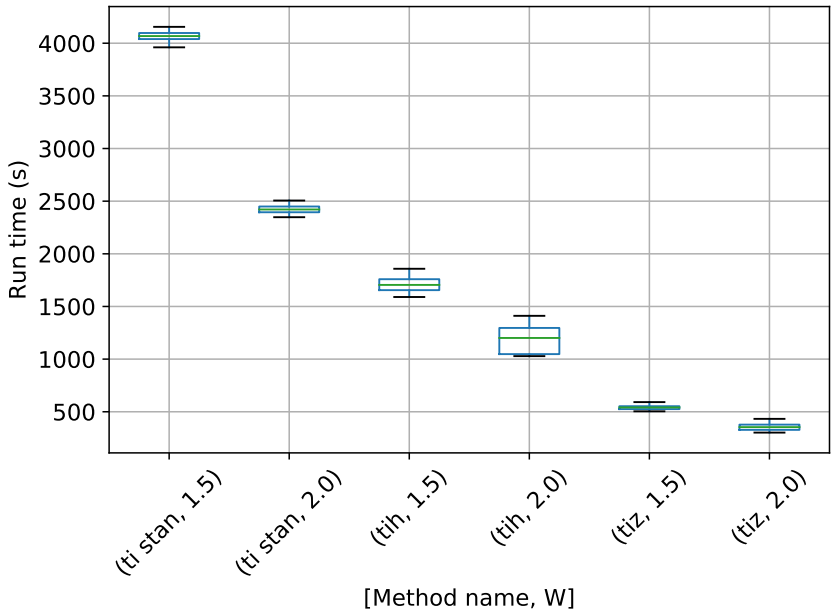


Figure 3.17: Box-plot of run time for the three stationary frequency model for TI-Stan, TI-BSS-H, and TI-BSS-Z, for two values of W

most versatile and robust, providing log-evidence estimates in a reasonable amount of time for a wide range of example problems. TI-BSS-H is also a robust technique, yielding accurate estimates for the example problems; however, TI-BSS-H has trouble yielding results in a timely manner when the number of dimensions increases beyond a certain point. TI-BSS-Z is less robust, not yielding results at all in many of the examples presented. For problems with less troublesome distributions, it may be worth considering as an option, especially because its run time tends to be significantly less than TI-BSS-H.

CHAPTER 4

ALGORITHM PERFORMANCE ANALYSIS

This chapter addresses the performance of the algorithms developed in Chapters 2 and 3, with respect to time complexity, precision, and accuracy. Theoretical and empirical analyses are presented for each technique.

4.1 Theoretical Time Complexity Analysis

This section contains theoretical analyses of each algorithm involved in the nested sampling and thermodynamic integration-based methods developed in Chapters 2 and 3. The algorithms are each listed in pseudo-code that emphasizes readability and elides certain implementation details that do not meaningfully affect either the time complexity or correctness of these algorithms. These pseudo-code implementations are shown in Chapters 2 and 3.

4.1.1 Serial nested sampling

The original (serial) nested sampling algorithm from Skilling [2006] is shown in Algorithm 2.1. For this algorithm, assuming a fixed log-likelihood function and MCMC function (as called on line 21), the only adjustable parameter is the number of live samples, N. I assume that the MCMC function is written in such a way as its time complexity is not a function of the likelihood threshold $minlogL$. I also assume that the number of parameters P is fixed based on the problem and that MIN and MAX are set such that they do not affect running time.

With this view, just a few lines of Algorithm 2.1 control the time complexity. Line 7 resides within two nested for loops, hence it runs NP times. Line 9 is only within the

outermost loop, so it runs N times. Lines 14 through 28 reside within a while loop, which, given my conditions on MIN and MAX, ends after $3HN$ iterations. H is fixed for a given problem, and each line within the while loop runs in constant time with respect to N, so the code within the while loop has time complexity $\mathcal{O}(N)$. The overall time complexity, omitting constant time parts and recalling that P is constant, is then

$$T(N) = \mathcal{O}(PN) + \mathcal{O}(N) = \mathcal{O}(N). \tag{4.1}$$

4.1.2 Multiple replacement nested sampling

The multiple replacement nested sampling algorithm [Burkoff et al., 2012, Henderson and Goggans, 2014] is shown in Algorithm 2.2. This algorithm is meant to be run with the loop in lines 20 and 21 parallelized, but a serial analysis is required before I can address the parallel portion. I carry forward my assumptions from analyzing the previous algorithm, except that here I assume that N and R are the parameters that may be varied.

Algorithm 2.2 is similar to Algorithm 2.1, with just a few modifications. The first notable difference occurs on line 14, where two arrays must be sorted. Assuming quicksort is used, line 14 has average time complexity $\mathcal{O}(N \log N)$ each time it is run. The second notable difference occurs on line 19, in which R random integers are chosen. Assuming a random integer can be chosen in constant time, this line takes $\mathcal{O}(R)$ each time it is run. Finally, line 21 is run R times per iteration of the while loop. The stopping condition for the main while loop in this algorithm is met when the counter reaches $3HN/R$. Hence, the overall time complexity for Algorithm 2.2 when running serially is

$$\begin{aligned} T(N,R) &= \mathcal{O}(PN) + (3HN/R) \times [\mathcal{O}(N \log N) + \mathcal{O}(R) + \mathcal{O}(R)] && (4.2)\\ &= \mathcal{O}(N) + \mathcal{O}(N^2 \log N/R) + \mathcal{O}(N) + \mathcal{O}(N) && (4.3)\\ &= \mathcal{O}(N) + \mathcal{O}(N^2 \log N/R). && (4.4) \end{aligned}$$

It would be natural to reduce (4.4) to $\mathcal{O}(N^2 \log N/R)$. However, due to the computational weight of the log-likelihood function, the constant on the $\mathcal{O}(N)$ portion, in practice, far exceeds $N \log N/R$, so the complexity is closer to $\mathcal{O}(N)$. This assumption is valid only for moderately sized values of N. For a sufficiently large N, the time complexity here would be $\mathcal{O}(N^2 \log N/R)$.

Before I compare the time complexity of Algorithm 2.2 to that of Algorithm 2.1, I need to consider the findings of [Henderson and Goggans, 2014]. Specifically, I need to consider that in order to maintain an equivalent level of variance in the evidence estimate as in a run of serial nested sampling with N live samples, a run of multiple replacement nested sampling discarding R samples at each likelihood contour must raise the number of live samples used to

$$N_R = \sqrt{R}N. \tag{4.5}$$

So, the time complexity of the multiple replacement nested sampling algorithm becomes $\mathcal{O}(N_R) = \mathcal{O}(\sqrt{R}N)$.

4.1.2.1 Parallel Speed-up

In the analysis of parallel algorithms, Amdahl's law provides an upper limit on the amount of speed-up that can be gotten from parallelizing an algorithm. If in a purely serial version of an algorithm, a portion that takes a fraction p of the total execution time can be parallelized, then the upper limit on the speed-up is

$$s = \frac{1}{1-p}. \tag{4.6}$$

For instance, if half of a program could be parallelized ($p = 0.5$), the upper limit on the speed-up would be 2.

In the case of nested sampling, for most problems, the vast majority of the execution time is spent in computing the log-likelihood function. For a fixed log-likelihood function, if more data or more parameters are used, the time spent computing the log-likelihood function

increases. Therefore, the Amdahl's law limit is not useful here.

A practical limit on the speed-up of the multiple replacement nested sampling algorithm is instead the value of R. As the log-likelihood's computation time increases, Algorithm 2.2 spends more of its time in lines 20, 21, and 22. Therefore, in the limit of an infinitely complex log-likelihood function, the upper limit of the speed-up of Algorithm 2.2 is R, for $R < N$, hence the lower limit on the parallel time complexity of Algorithm 2.2 is

$$T(N, R) = \mathcal{O}(N/\sqrt{R}). \tag{4.7}$$

Finally, comparing (4.7) to (4.1), the best case speed-up for parallel multiple replacement nested sampling over serial nested sampling is $\mathcal{O}(N)/\mathcal{O}(N/\sqrt{R}) = \mathcal{O}(\sqrt{R})$.

4.1.3 Combined chain nested sampling

The combined-chain nested sampling algorithm [Henderson et al., 2017] is shown in Algorithm 2.3. This procedure runs the serial nested sampling procedure (Algorithm 2.1) M times, then combines and sorts the results, and finally recomputes the log-evidence based on the combined results. The parameters of interest to the user for tuning here are N and M. In lines 3, 4, and 5, nested sampling, which runs in $\mathcal{O}(N)$ time, is run M times, for a total time complexity of $\mathcal{O}(MN)$. Each nested sampling run should return $\mathcal{O}(N)$ total samples, for an overall total of $\mathcal{O}(MN)$ samples. Hence, the sort function in line 6 runs in $\mathcal{O}(MN \log(MN))$ time. Each line within the for loop that starts at line 10 takes constant time, so the loop overall runs in $\mathcal{O}(MN)$ time. For the procedure, the overall time complexity is then

$$T(N, M) = \mathcal{O}(MN) + \mathcal{O}(MN \log(MN)) + \mathcal{O}(MN). \tag{4.8}$$

As with multiple-replacement nested sampling, here is another case where the constant on the first $\mathcal{O}(MN)$ portion is far larger than $\log(MN)$, so for realistic problems sizes

and values of M and N, the overall time complexity of Algorithm 2.3 is

$$T(N, M) = \mathcal{O}(MN), \tag{4.9}$$

which will be verified empirically in Section 4.2.

Chapter 2 establishes that the error in the log-evidence estimate is equivalent for a single serial nested sampling run using $N_M = NM$ live samples as in a combined nested sampling run using M runs each with N live samples, provided that the constrained prior can be sampled sufficiently well in each case. A serial nested sampling run with these parameters would run in $\mathcal{O}(MN)$ time. A serial implementation of combined-chain nested sampling with these parameters would also run in $\mathcal{O}(MN)$ time.

4.1.3.1 Parallel Speed-up

Here again as in Algorithm 2.2, the Amdahl's Law limit is usually not useful. The value of M provides a hard limit on how many independent workers can be used effectively, and as M increases, a greater percentage of the code can be made parallel. To find the upper limit on the parallel speed-up between combined-chain nested sampling and serial nested sampling, I again consider the scenario in which serial nested sampling is run with NM live samples, and combined-chain nested sampling is run with M independent nested sampling processes each with N live samples. Assuming at least M workers are available, and noting that each iteration of line 4 should take approximately the same amount of time, the loop in lines 3, 4, and 5 runs in $\mathcal{O}(N)$ time. This loop occupies the vast majority of the actual run time of the program, so the time complexity overall for Algorithm 2.3 becomes $\mathcal{O}(N)$. The best-case speed-up compared to serial nested sampling is then $\mathcal{O}(MN)/\mathcal{O}(N) = \mathcal{O}(M)$.

4.1.4 Thermodynamic integration with binary slice sampling

The thermodynamic integration algorithm with importance sampling with re-sampling, adaptive annealing, and binary slice sampling is shown in Algorithm 3.2. Several auxiliary

functions are required for this algorithm, including binary slice sampling (Algorithm 3.3), leapfrog sampling (Algorithm 3.4), importance sampling with re-sampling (Algorithm 3.1), and a line-to-axes function (either Algorithm 3.5 or Algorithm 3.6). An axes-to-line function is also required, but it is not shown here, as it is nearly exactly the reverse of the line-to-axes function. This procedure can use either Hilbert or Z-order space filling curves to accomplish the line-to-axes and axes-to-line moves. Both Hilbert and Z-order curve functions are given here.

The time complexity of the main thermodynamic integration procedure depends on the time complexity of its subordinate functions, so the analyses of those functions is presented first.

4.1.4.1 Hilbert curve line-to-axes function

This algorithm is shown in Algorithm 3.5. For time complexity analysis of this function, B, the number of bits per parameter, and P, the number of parameters, are the parameters of interest. This analysis assumes that bit operations run in constant time with respect to these parameters.

The loop in lines 2 through 4 and the loop in lines 6 through 8 each run several constant-time operations P times, so those sections run in $\mathcal{O}(P)$ time. The for loop starting at line 10 contains a while loop, which depends on the value of j. j is initialized as M, which is 1 bitshifted to the left by $B-1$, which is equivalent to 2^{B-1}. So, j starts out at 2^{B-1} and is bitshifted to the right by 1 at each iteration of the while loop until it reaches 0. j reaches 0 after B right bitshifts, so the while loop runs B times. Therefore, the nested for-while loops from line 10 to 22 run in $\mathcal{O}(PB)$ time.

There is another for loop that runs a constant-time expression $P-1$ times on lines 24 through 26, so that loop runs in $\mathcal{O}(P)$ time. The while loop that starts at line 28 runs until Q is equivalent to M. Q starts at 2, and is bitshifted to the left by 1 on every iteration of the while loop. Again, M is 2^{B-1}, so Q will need to be shifted left $B-2$ times to reach 2^{B-1}. Hence, the while loop starting on line 28 will run $B-2$ times. The for loop within

this while loop runs $P-1$ times, with only constant time-expressions within it. Therefore, the nested while-for loops from line 28 to 41 run in $\mathcal{O}(PB)$ time.

The overall time complexity of Algorithm 3.5 is then

$$T(B,P) = 3\mathcal{O}(P) + 2\mathcal{O}(PB) = \mathcal{O}(PB). \tag{4.10}$$

4.1.4.2 Z-order curve line-to-axes function

This algorithm is shown in Algorithm 3.7. As with the Hilbert curve function, the parameters of interest here are B and P. While this function does depend on the mask computation function shown in Algorithm 3.6, the mask function only needs to be run once for each combination of B and P, so the time complexity of Algorithm 3.7 does not depend on the time complexity of Algorithm 3.6. However, the number of masks that are generated in Algorithm 3.6 is important for finding the time complexity of Algorithm 3.7. Line 33 of that algorithm is responsible for appending masks to the main list of masks that gets returned. The number of times that line will be run is not an immediately obvious function of B and N, but the upper limit on is the length of the *moveBits* list. The *moveBits* list is built on lines 9 and 13, and the upper limit on its length is *maxLength*. *maxLength* is set as the length of the longest string in *bd*, which would be the bit string corresponding to $(B+1)(P-1)$, with the leading two bits removed. So, the upper limit on the value of *maxLength* is $\log_2[(B+1)(P-1)] - 2 = \log_2(B+1) + \log_2(P-1) - 2$. Hence, there are $\mathcal{O}(\log B + \log P)$ masks generated.

Returning to Algorithm 3.7, the first line with non-constant time complexity is line 7, which in most implementations will run linear time for the size of the list. The list of masks has length $\mathcal{O}(\log B + \log P)$, so this line runs in $\mathcal{O}(\log B + \log P)$ time. The for loop starting at line 10 runs P times, and the inner for loop starting at line 14 runs $\mathcal{O}(\log B + \log P)$ times. The time complexity of the nested for loops is therefore $P\mathcal{O}(\log B + \log P)$. The

overall time complexity of the algorithm is then

$$T(B,P) = \mathcal{O}(\log B + \log P) + \mathcal{O}(P\log B + P\log P) \tag{4.11}$$

$$= \mathcal{O}(P\log B + P\log P) \tag{4.12}$$

4.1.4.3 Binary slice sampling

This algorithm is shown in Algorithm 3.3. The parameters of interest here are the number of bits per dimension B_{in}, the number of parameters P, and the number of Monte Carlo steps S. Line 4 presents the first non-constant time expression, with a call to the line-to-axes function. Depending on whether the Hilbert or Z-order curves are in use, this line runs in either $\mathcal{O}(PB)$ or $\mathcal{O}(P\log B + P\log P)$ time. The for loop that begins on line 6 runs S times. The lines within the if block starting on line 7 run in the worst case as many times as the for loop, with a time complexity of either $\mathcal{O}(PB)$ or $\mathcal{O}(P\log B + P\log P)$. The invocation of the axes-to-line function on line 12 runs in either $\mathcal{O}(PB)$ or $\mathcal{O}(P\log B + P\log P)$ time. On line 14, the energy function is called, which has an unknown time complexity that may depend on P, so it is denoted $f(P)$. In the worst case, the while loop starting on line 17 runs B_{in} times. Each line within this loop runs in constant time, except the line-to-axes call on line 20, which runs in either $\mathcal{O}(PB)$ or $\mathcal{O}(P\log B + P\log P)$ time, and the energy function on line 22, which runs in $f(P)$ time.

For the Hilbert curve case, the overall worst-case time complexity is

$$T(S,B_{in},P) = \mathcal{O}(PB_{in}) + S\{f(P) + \mathcal{O}(PB_{in}) + \mathcal{O}(PB_{in}) + B_{in}[\mathcal{O}(PB_{in}) + f(P)]\} \tag{4.13}$$

$$= \mathcal{O}[Sf(P)] + \mathcal{O}(SB_{in}P) + \mathcal{O}(SB_{in}^2P) + \mathcal{O}[SB_{in}f(P)] \tag{4.14}$$

$$= \mathcal{O}(SB_{in}^2P) + \mathcal{O}[SB_{in}f(P)]. \tag{4.15}$$

For the Z-order curve case, the overall worst-case time complexity is

$$T(S, B_{in}, P) = \mathcal{O}(SB_{in}P \log B_{in} + SB_{in}P \log P) + \mathcal{O}[SB_{in}f(P)]. \tag{4.16}$$

4.1.4.4 Leapfrog sampling

This algorithm is shown in Algorithm 3.4. The parameters of interest here are the number of bits per dimension B, the number of chains C, and the number of parameters P. Let $f(P)$ be the time complexity of the energy function, and let $g(B, P)$ be the time complexity of the line-to-axes and axes-to-line functions. The loop that starts at line 2 runs C times, with an overall time complexity of $Cg(B, P)$. Assuming quicksort is used, line 5 runs in average time of $\mathcal{O}(C \log C)$. The loop that starts in line 6 runs C times, line 7 runs in $g(B, P)$ time per iteration, and line 8 runs in $f(P)$ time per iteration, for an overall loop time complexity of $\mathcal{O}\{C[g(B, P) + f(P)]\}$. The for loop that starts on line 10 runs C times. Lines 22, 23, and 25 run in $g(B, P)$ time, and, assuming each trial is accepted, line 27 is run at each iteration of the loop with time complexity $f(P)$. The overall time complexity of this algorithm is

$$T(C, B, P) = \mathcal{O}[Cg(B, P)] + \mathcal{O}(C \log C) + \mathcal{O}\{C[g(B, P) + f(P)]\} + \mathcal{O}\{C[3g(B, P) + f(P)]\} \tag{4.17}$$

$$= \mathcal{O}[Cg(B, P) + Cf(P)] + \mathcal{O}(C \log C). \tag{4.18}$$

For the Hilbert curve case, this becomes

$$T(C, B, P) = \mathcal{O}[CBP + Cf(P)] + \mathcal{O}(C \log C), \tag{4.19}$$

and for the Z-order curve case, it becomes

$$T(C, B, P) = \mathcal{O}[CP \log B + CP \log P + Cf(P) + C \log C]. \tag{4.20}$$

4.1.4.5 Importance sampling with re-sampling

This algorithm is shown in Algorithm 3.1. The only parameter of interest here is the number of chains, C. There are three sort operations which take place on line 2, which run in an overall average time of $\mathcal{O}(C \log C)$, assuming quicksort is used. The sum in line 3 takes $\mathcal{O}(C)$ time. The cumulative sum in lines 6 through 8 runs in $\mathcal{O}(C \log C)$ time. After the normalization that happens in line 3, w sums to C. Therefore, after the cumulative sum in lines 6 through 8, the final element of w is C. The two nested loops at 11 and 12 interact in such a way that the lines from 13 to 17 run only C times. Each time the while loop iterates, line 16 runs, which increments u by 1. Once the while loop has iterated C times, u will be at least C, which means the while loop will no longer iterate. Hence, the nested loops run in $\mathcal{O}(C)$ time. The overall time complexity of the algorithm is

$$T(C) = \mathcal{O}(C \log C) + \mathcal{O}(C) + \mathcal{O}(C \log C) + \mathcal{O}(C) \tag{4.21}$$

$$= \mathcal{O}(C \log C) \tag{4.22}$$

4.1.4.6 Main TI procedure

This algorithm is shown in Algorithm 3.2. The parameters of interest are the number of parameters P, the number of chain steps M, the number of BSS steps S, the number of chains C, the number of bits per dimension B, and the adaptive annealing control parameter W. The lines within the for loop starting at line 3 run C times each. Line 5 runs in $g(B, P)$ time, while line 6 runs in $f(P)$ time. Computing the expectation in line 9 has time complexity $\mathcal{O}(C)$ The inner max and min operations in line 10 run in $\mathcal{O}(C)$ time. The importance sampling invocation at line 12 runs in $\mathcal{O}(C \log C)$ time.

Because the temperature schedule is adaptive and very much dependent on the energy function, we cannot predict ahead of time how many times the main while loop that begins on line 13 will iterate. While the total number of iterations is unknown ahead of time, it is known that smaller values of W and larger values of C cause the steps in β to become smaller, leading

to more iterations of the loop. Hence, we denote the number of iterations of the main while loop as $h(C, W)$. Due to the nested for loops beginning at lines 14 and 15, line 16 runs MC times, and itself has time complexity $\mathcal{O}[SBg(B, P)+SBf(P)]$. Line 18 is within only one for loop, so it runs M times with a time complexity of $\mathcal{O}[CP\log B+CP\log P+Cf(P)C\log C]$. Line 21 contains one max and one min operation, which together have time complexity $\mathcal{O}(C)$. Line 27 has time complexity $\mathcal{O}(C\log C)$. Finally, line 29 needs to loop through each value of β and $\langle E\rangle$, so it has time complexity $\mathcal{O}[h(C, W)]$. The overall time complexity of the algorithm is

$$\begin{aligned} T(P, M, S, C, B, W) = {} & C\mathcal{O}[g(B, P) + f(P)] + 2\mathcal{O}(C) + \mathcal{O}(C\log C)+ \\ & h(C, W)\{MC\mathcal{O}[SBg(B, P) + SBf(P)] + M\mathcal{O}[Cg(B, P) + Cf(P) + C\log C]+ \\ & \mathcal{O}(C) + \mathcal{O}(C\log C)\} + \mathcal{O}(C), \end{aligned} \quad (4.23)$$

which can be simplified to

$$T(P, M, S, C, B, W) = \mathcal{O}\{h(C, W)MC[SBg(B, P) + SBf(P) + \log C]\}. \quad (4.24)$$

If the Hilbert curve is used, this becomes

$$T(P, M, S, C, B, W) = \mathcal{O}\{h(C, W)MC[SB^2P + SBf(P) + \log C]\}, \quad (4.25)$$

while if the Z-order curve is used, it becomes

$$T(P, M, S, C, B, W) = \mathcal{O}\{h(C, W)MC[SBP\log B+SBP\log P+SBf(P)+\log C]\}. \quad (4.26)$$

For a given problem, we cannot actually control P. Without that method parameter, the

time complexity with the Hilbert curve becomes

$$T(M, S, C, B, W) = \mathcal{O}\{h(C, W)MC[SB^2 + \log C]\}, \tag{4.27}$$

while the time complexity with the Z-order curve becomes

$$T(M, S, C, B, W) = \mathcal{O}\{h(C, W)MC[SB \log B + \log C]\}. \tag{4.28}$$

4.1.4.7 Parallel speed-up of TI with BSS

The portion of Algorithm 3.2 that can be readily run in parallel is contained in the for loop in lines 15 through 17. This the invocation of binary slice sampling within this loops happens once per chain, and each run is independent of the others. The number of chains C tends to be on the order of 100, and for a consumer-level computer, that is likely more than the number of processors present in the machine. Here, the Amdahl's law limit or the number of processors in the machine might limit how much the algorithm can benefit from parallelism. The following is the upper limit on parallel speed-up in the case that neither the Amdahl's law limit or hardware limits are in effect.

If each step of the binary slice sampling for loop can be run simultaneously, the time complexity of that loop becomes simply the time complexity of one run of binary slice sampling: $M\mathcal{O}[SBg(B, P)+SBf(P)]$. Under these assumptions, the overall time complexity of thermodynamic integration with the Hilbert curve becomes

$$T(M, S, C, B, W) = \mathcal{O}\{h(C, W)M[SB^2 + \log C]\}, \tag{4.29}$$

and the version with the Z-order curve becomes

$$T(M, S, C, B, W) = \mathcal{O}\{h(C, W)M[SB \log B + \log C]\}. \tag{4.30}$$

The speed-up in both cases is $\mathcal{O}(C)$.

4.1.5 Note about TI with Stan

The TI with Stan method described in Chapter 3 makes use of Stan to replace the binary slice sampling and leapfrog sampling methods. Since neither of these methods is used, the space-filling curve is also not necessary. Stan is a black box as far as analysis goes, and without a time-consuming exploration of the source code, an analytic time complexity cannot be presented for TI with Stan. This limitation aside, section 4.2 does explore an empirical analysis of TI with Stan's time complexity.

4.2 Empirical Time Complexity Analysis

This section examines the empirical time complexity and empirical parallel speed-up of the following algorithms: combined-chain nested sampling (Algorithm 2.3), multiple-replacement nested sampling (Algorithm 2.2), TI-BSS with the Hilbert curve (Algorithm 3.2), and TI-Stan (Algorithm 3.8). Figures summarizing run times for multiple runs of these algorithms over a range of parameter values are presented, and curves are fitted to these data, with commentary given on how well the analytical analyses in Section 4.1 match with the observed behavior. The multiple stationary frequency detection data used in Chapters 2 and 3 is used to perform this analysis. Here, I specifically use the model that assumes two stationary signals are present.

These tests were run on one of two Google Cloud instances set up specifically for generating these results. Each instance has 32 virtual CPUs and 28.8 GB of RAM. One instance uses Intel Haswell processors, while the other uses Intel Broadwell processors. The nested sampling speed-up tests and the TI parameter tests were done on the Haswell instance, while the TI speed-up and nested sampling parameter tests were done on the Broadwell instance. For a given chart in this section, the same instance was used for each run on that chart.

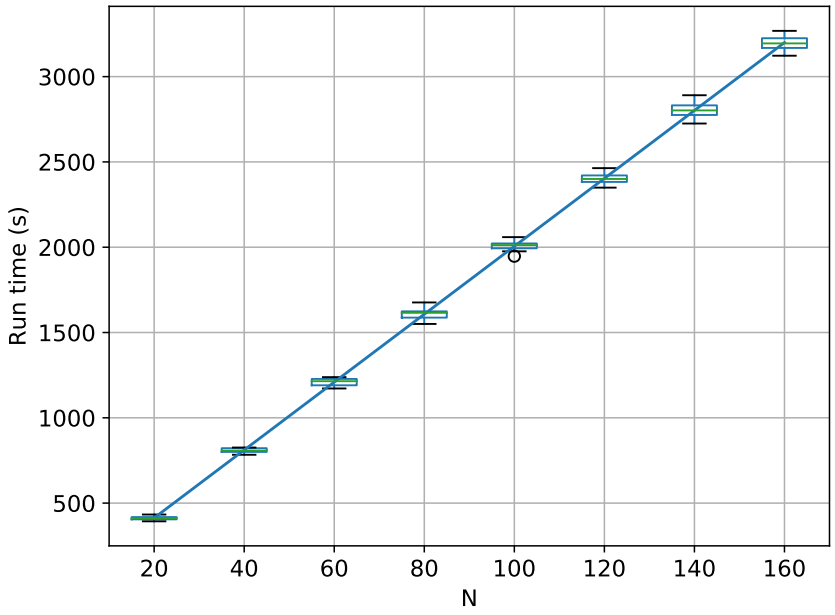


Figure 4.1: Box-plot for run time of combined-chain nested sampling vs N over 20 runs with a linear fitted curve

4.2.1 Nested Sampling

This subsection looks at the performance of nested sampling, both in the combined-chain and multiple replacement configurations. In Section 4.1, the analysis of combined-chain nested sampling predicts that the algorithm should be linear in the number of live samples N. The results below use $M = 32$ simultaneous nested sampling runs. Figure 4.1 shows a box-plot for the run time plotted against the number of of live samples over 20 runs and a linear fitted curve. The curve shown is

$$T(N) = 19.90N + 15.03. \tag{4.31}$$

Qualitatively, the linear curve fits well, supporting my initial analysis.

Figure 4.2 shows a box-plot for the log-evidence of combined-chain nested sampling plotted against the number of live samples over 20 runs. The precision in the log-evidence result increases drastically from 20 to 80, but remains steady after that. This suggests that

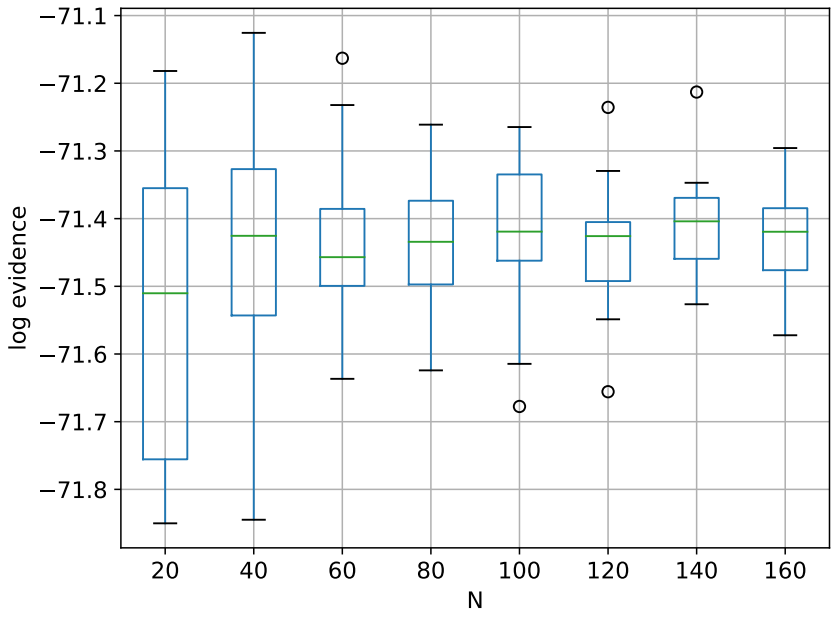


Figure 4.2: Box-plot for log-evidence of combined-chain nested sampling vs N over 20 runs

the remaining variation in the log-evidence estimate is dominated by a source other than the error in estimating the prior mass at each likelihood threshold.

The analysis in the Section 4.1 finds that combined-chain nested sampling should have a parallel speed-up that is linear with M, hence

$$T(M) = \mathcal{O}(1/M). \tag{4.32}$$

Figure 4.3 shows a box-plot for the run time plotted against the number of simultaneous runs of nested sampling over 20 runs and a $1/M$ fitted curve. The number of live samples starts at 512 for $M = 1$ and is halved for every increase of M until it is 32 for $M = 16$. The fitted curve in Figure 4.3 is

$$T(M) = \frac{7192}{M} + 26.79. \tag{4.33}$$

The curve appears to fit the results well.

Figure 4.4 shows a box-plot for the log-evidence of combined-chain nested sampling plotted against the number of simultaneous runs of nested sampling over 20 runs. Figure

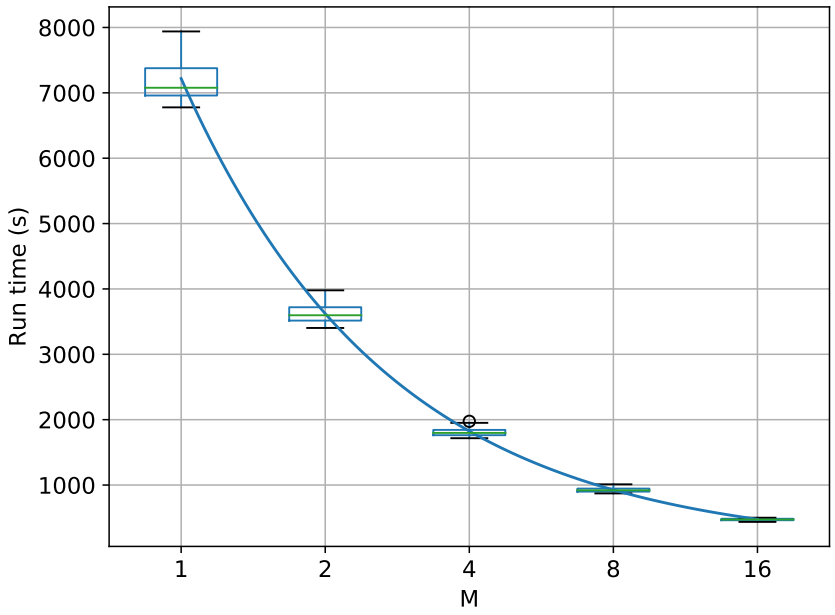


Figure 4.3: Box-plot for run time of combined-chain nested sampling vs M over 20 runs with a 1/M fitted curve

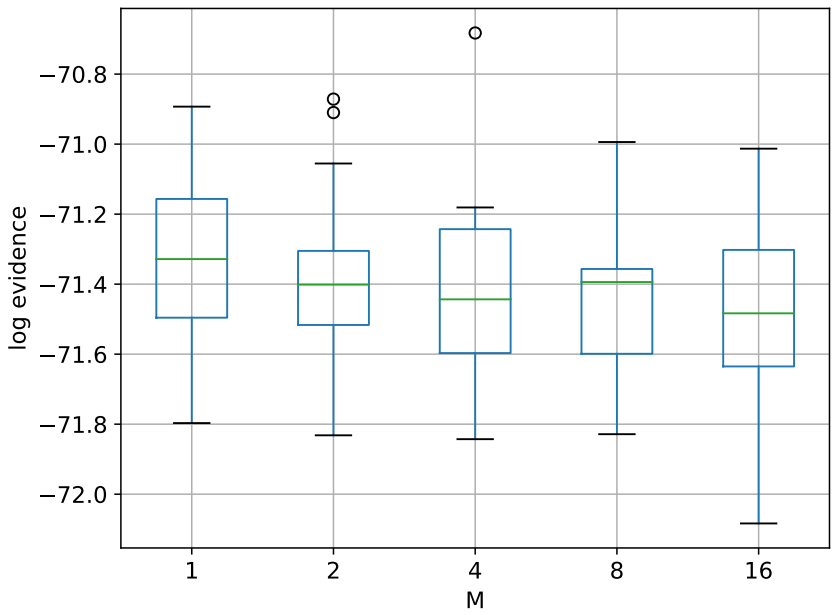


Figure 4.4: Box-plot for log-evidence of combined-chain nested sampling vs M over 20 runs

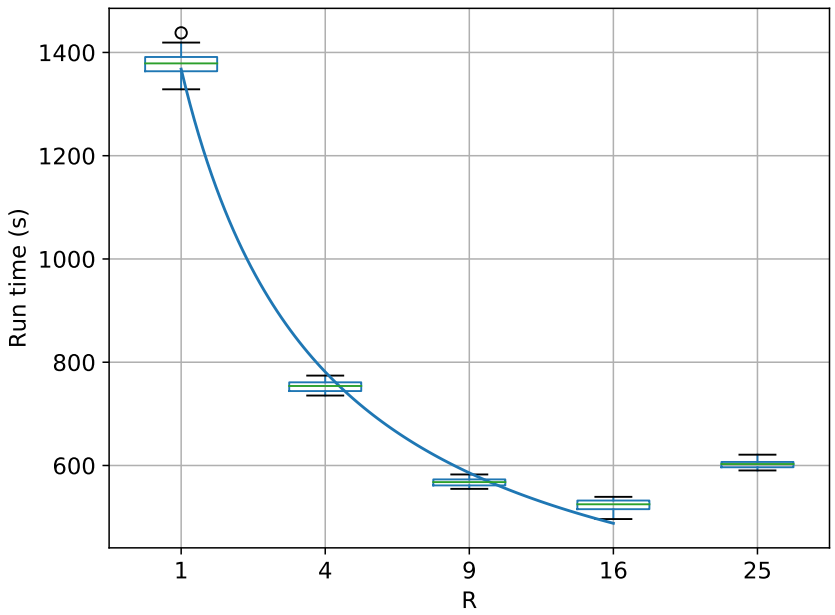


Figure 4.5: Box-plot for run time of multiple-replacement nested sampling vs R over 20 runs with a $1/\sqrt{R}$ fitted curve

4.4 shows that the precision of the log-evidence remains roughly constant with the change in M, as we would expect (keeping in mind that N is reduced as M is increased such that MN remains constant).

In contrast to combined-chain nested sampling, multiple-replacement nested sampling should have a $\sqrt{R}$ parallel speed-up. Figure 4.5 shows a box-plot for the run time plotted against the number of replaced samples at each likelihood threshold over 20 runs and a $1/\sqrt{R}$ fitted curve. The number of live samples starts at $N = 100$ and increases as $\sqrt{R}$, ending at $N = 500$. The fitted curve in Figure 4.5 is

$$T(R) = \frac{1173}{\sqrt{R}} + 194.7. \tag{4.34}$$

The curve fits fairly well until $R = 25$, at which point the run time actually increases. A profiler would be necessary to truly diagnose this issue, but it can be reasonably assumed that the overhead costs of the algorithm as N approaches high levels (500) become larger than the gain from shortening the overall number of runs required to complete the evidence

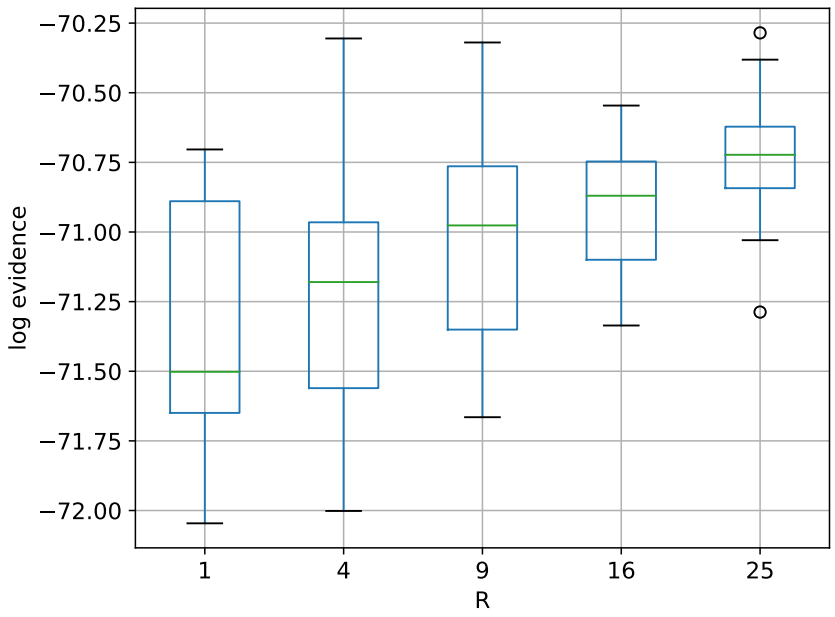


Figure 4.6: Box-plot for log-evidence of multiple-replacement nested sampling vs R over 20 runs

estimate. Recall that the multiple-replacement nested sampling algorithm requires a full sort of the live samples at each likelihood threshold, so high values of N could potentially introduces a large amount of overhead.

Figure 4.6 shows a box-plot for the log-evidence of multiple-replacement nested sampling plotted against the number of replaced samples at each likelihood threshold over 20 runs. The precision of the log-evidence estimate actually seems to improve as R (and thus N) increases. The error in the log-evidence due to error in the prior mass estimate should not change, but the sampling of the constrained prior at each likelihood threshold likely becomes more effective with increase N, explaining the increase in the precision of the log-evidence estimate.

4.2.2 Thermodynamic Integration with Stan

This subsection examines the performance of TI-Stan as several parameters are varied, and it examines the parallel speed-up of TI-Stan as more workers are made available. For these tests, unless otherwise specified, the parameters are set as shown in Table 4.1.

Table 4.1: Parameters for empirical analysis of TI-Stan

Parameter	Value	Definition
W	1.5	Constant to control annealing
S	200	Number of steps allowed in Stan
C	64	Number of chains

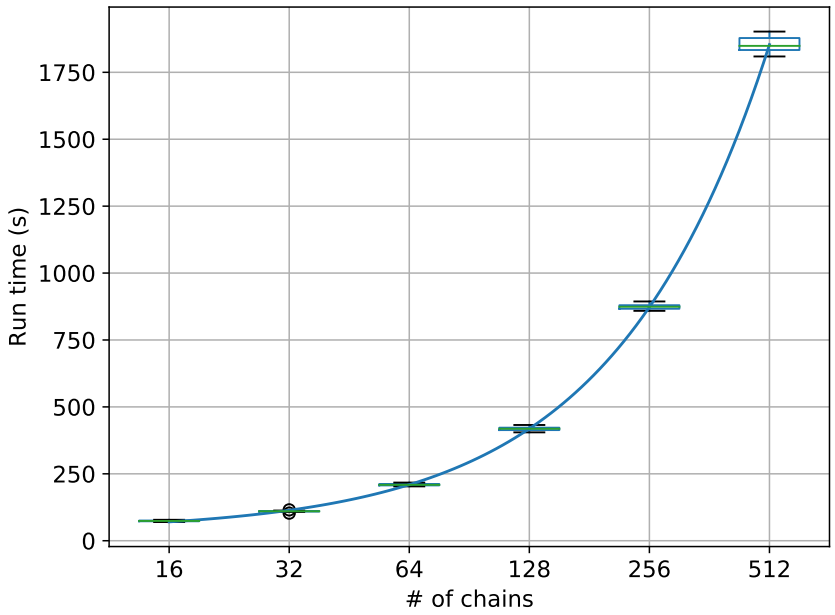


Figure 4.7: Box-plot for run time of TI-Stan vs C over 20 runs with a $C \log C$ fitted curve

The analysis in Section 4.1 finds that the time complexity of TI with respect to the number of chains should be $\mathcal{O}(C \log C)$. Figure 4.7 shows a box-plot for the run-time plotted against the number of chains and a $C \log C$ fitted curve. The fitted curve takes the form

$$T(C) = 0.41C \log C + 1.02C + 36.24, \tag{4.35}$$

confirming our expectations.

Figure 4.8 shows a box-plot for the log-evidence for TI-Stan plotted against the number of chains. As the number of chains increases, the precision in the log-estimate improves. It is also worth noting that the upper end of the range of results does not change very much,

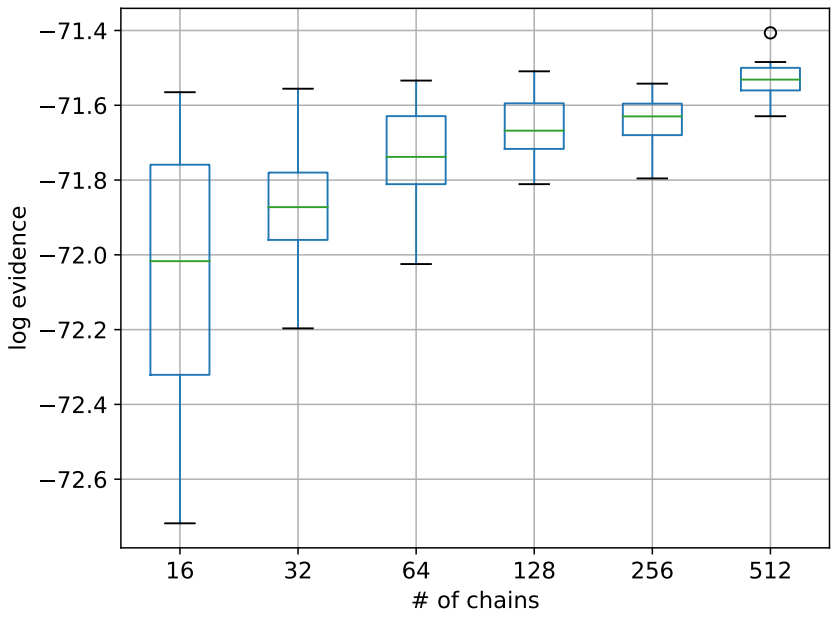


Figure 4.8: Box-plot for log-evidence of TI-Stan vs C over 20 runs

but the improvement in precision is seen in the changing of the lower end of the range.

The analysis in Section 4.1 suggests that the time complexity of TI with respect to the number of chain refresh iterations should be linear. Figure 4.9 shows a box-plot for the run-time plotted against the number of iterations allowed in Stan at each temperature and a linear fitted curve. The fitted curve takes the form

$$T(S) = 0.78S + 52.59, \tag{4.36}$$

which matches up with the analytical prediction. The fitted curve omits the results from $S = 20$ and the outliers at $S = 40$, $S = 100$ and $S = 150$. The fact that these values of S produce run times far greater than the other values suggest that there is an issue in the adaptive sampling conducted by Stan when enough iterations are not permitted.

Figure 4.10 shows a box-plot for the log-evidence for TI-Stan plotted against the number of Stan iterations per temperature. The estimated log-evidence is very similar across most of the values of S, except for $S = 20$. Here, the log-evidence appears to be significantly

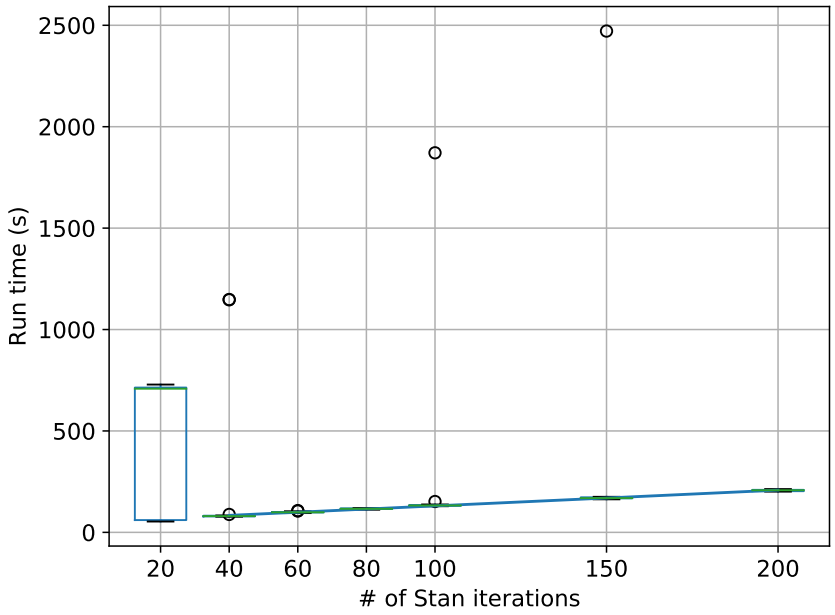


Figure 4.9: Box-plot for run time of TI-Stan vs S over 20 runs with a linear fitted curve

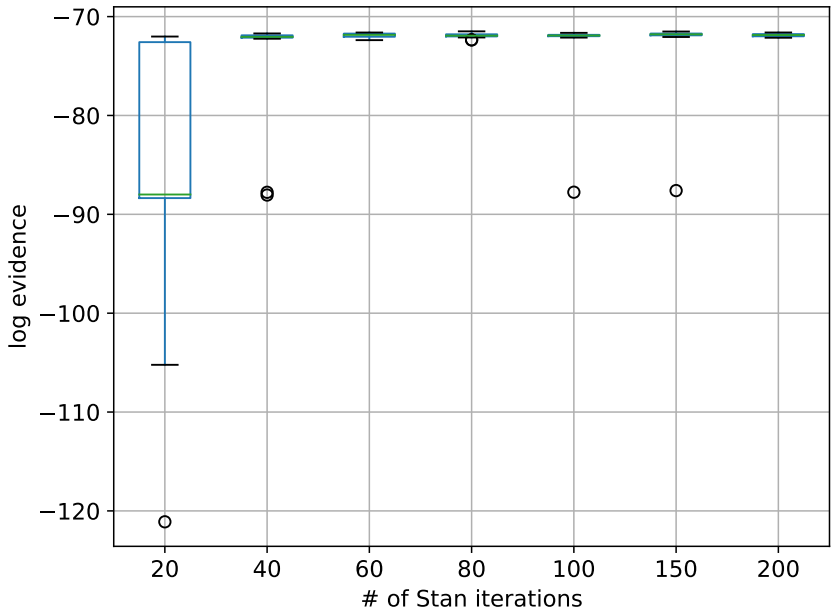


Figure 4.10: Box-plot for log-evidence of TI-Stan vs S over 20 runs

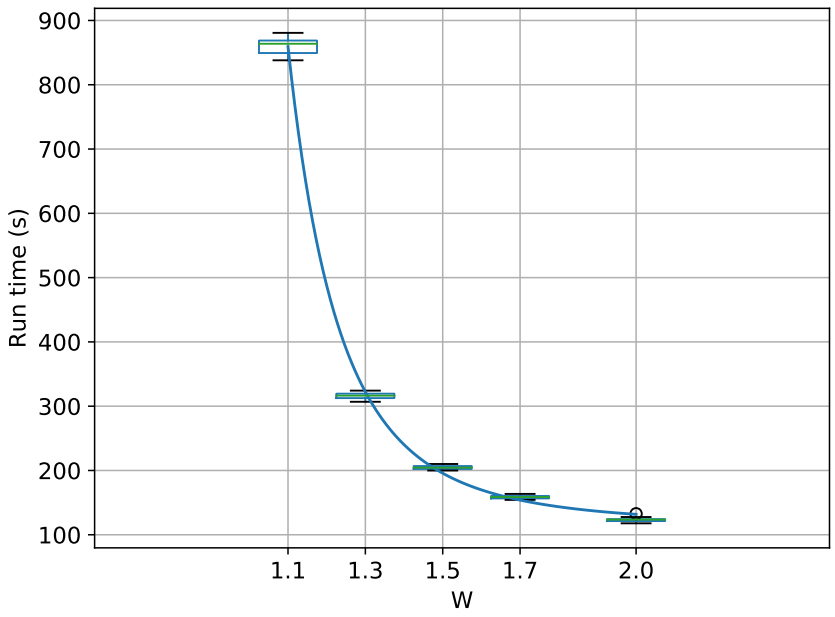


Figure 4.11: Box-plot for run time of TI-Stan vs W over 20 runs with an exponential fitted curve

under-estimated in most of the runs. There are also a couple of very low-log-evidence outliers with $S = 40$. This suggests that for a given problem, there is a lower limit on the number of iterations Stan needs to properly refresh the population of samples, but after this limit is reached, no significant gains in precision are achieved.

The analysis in Section 4.1 notes that TI's time complexity is probably a function of the adaptive annealing control parameter W, but the precise relationship is unknown. Figure 4.11 shows a box-plot for the run-time plotted against the value of the adaptive annealing control parameter and an exponential fitted curve. The fitted curve takes the form

$$T(W) = 0.19 \exp(9.11/W) + 113.83. \tag{4.37}$$

The adaptive annealing in TI is set up such that if W were set to 1, the temperature would never change, and the program would run forever. The function in (4.37) is appropriate in this context, and fits the data well. Figure 4.11 emphasizes an unfortunate relationship that exists within TI: a value of W closer to 1 yields a more accurate result, but takes

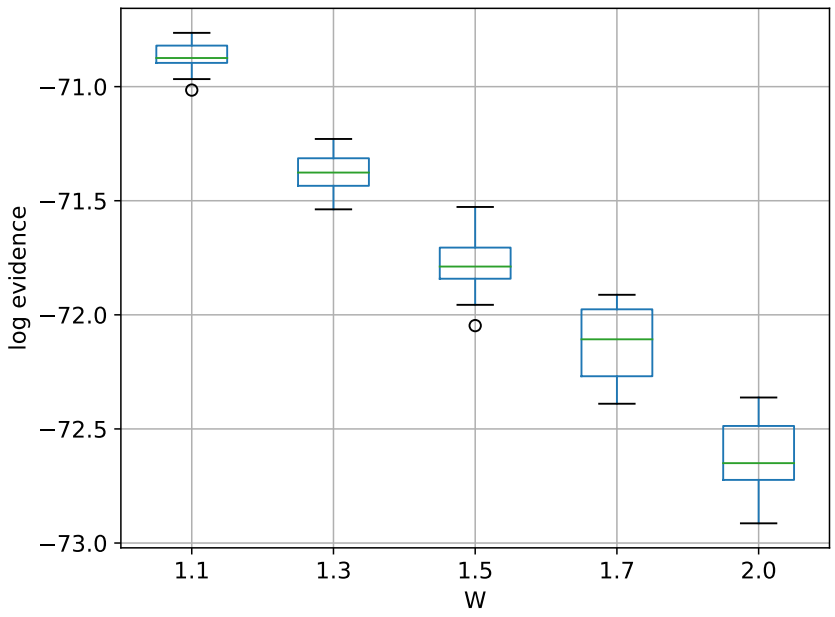


Figure 4.12: Box-plot for log-evidence of TI-Stan vs W over 20 runs

exponentially more time.

Figure 4.12 shows a box-plot for the log-evidence for TI-Stan plotted against the adaptive annealing control parameter. As W approaches 1, the log-evidence both increases somewhat and has a higher precision over the runs. This is consistent with previous observations about W's effect on the log-evidence estimate.

The analysis in Section 4.1 suggests that the speed-up of TI should be linear with the number of workers, possibly up to the number of chains. Figure 4.13 shows a box-plot for the run-time plotted against the number of available workers for a fixed value of $C = 64$ with a fitted $1/N$ curve. The fitted curve takes the form

$$T(N) = \frac{1941}{N} + 48.05. \tag{4.38}$$

Considering the observations from Figure 4.8 that a greater value of C yields a higher precision in the log-evidence estimate, the linear speed-up observed with up to 32 workers confirms that this technique can be made highly parallel to great benefit.

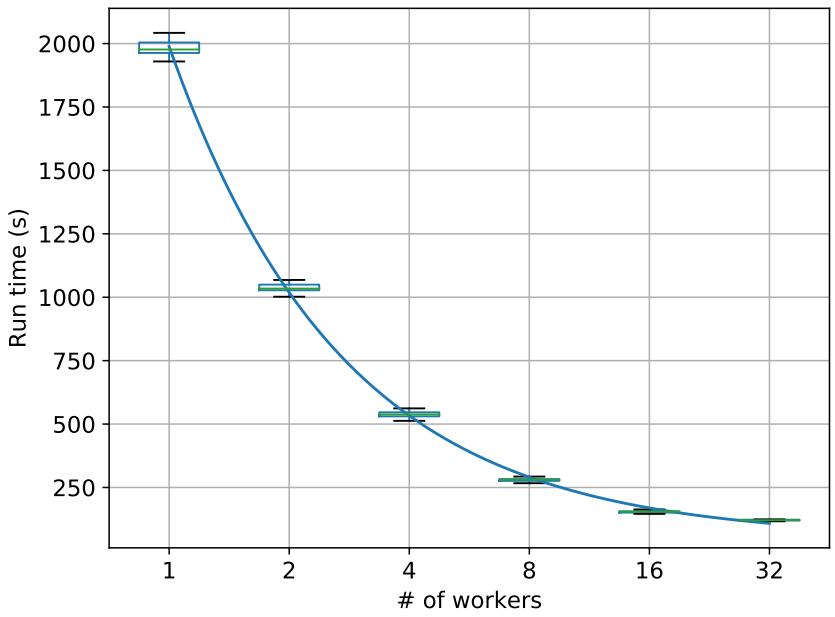


Figure 4.13: Box-plot for run time of TI-Stan vs the number of workers over 20 runs with a $1/N$ fitted curve

Figure 4.14 shows a box-plot for the log-evidence for TI-Stan plotted against the number of available workers. The number of workers should not affect the precision or accuracy of the result, and as expected there is no systematic change observed in the log-evidence results as the number of workers changes.

4.2.3 Thermodynamic Integration with Binary Slice Sampling and the Hilbert Curve

This subsection examines the performance of TI-BSS-H as several parameters are varied, and it examines the parallel speed-up of TI-BSS-H as more workers are made available. For these tests, unless otherwise specified, the parameters are set as shown in Table 4.2.

The results in this subsection for parameters shared with TI-Stan track closely with the results for TI-Stan.

The analysis in Section 4.1 finds that the time complexity of TI with respect to the number of chains should be $\mathcal{O}(C \log C)$. Figure 4.15 shows a box-plot for the run-time plotted against the number of chains and a $C \log C$ fitted curve. The fitted curve takes the

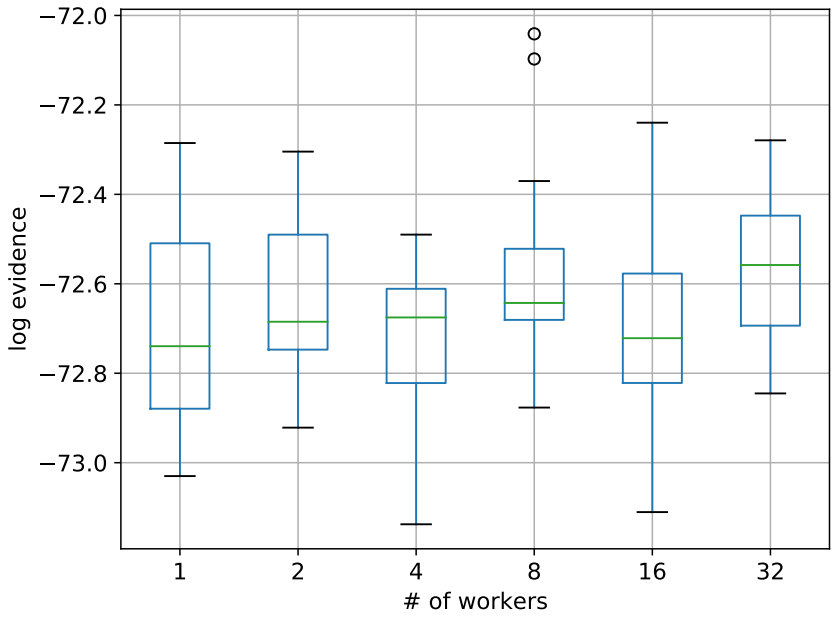


Figure 4.14: Box-plot for log-evidence of TI-Stan vs the number of workers over 20 runs

Table 4.2: Parameters for empirical analysis of TI-BSS-H

Parameter	Value	Definition
W	1.5	Constant to control annealing
S	200	Number of binary slice sampling steps
M	2	Number of combined binary slice sampling and leapfrog steps
C	64	Number of chains
B	32	Number of bits per parameter in SFC

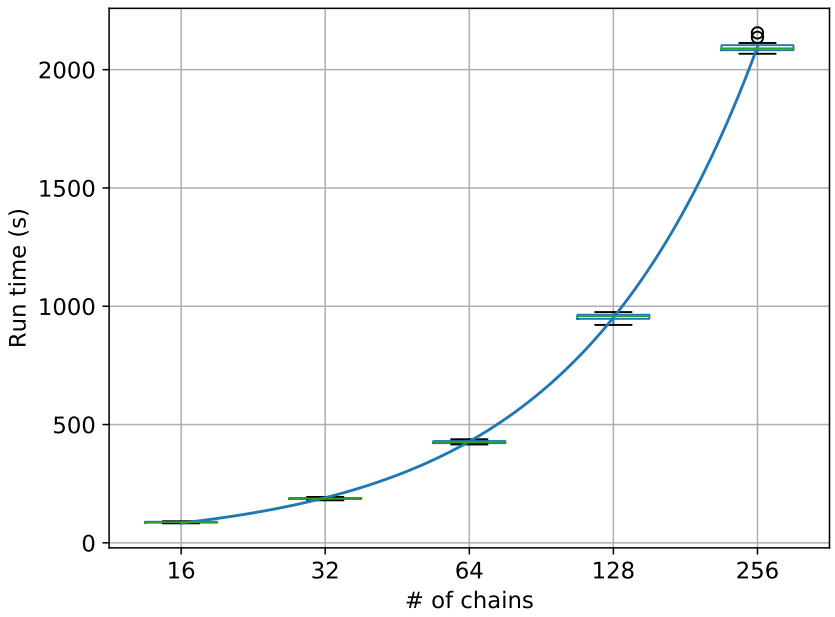


Figure 4.15: Box-plot for run time of TI-BSS-H vs C over 20 runs with a $C \log C$ fitted curve

form

$$T(C) = 1.09C \log C + 2.16C + 1.09, \tag{4.39}$$

which matches our expectations.

Figure 4.16 shows a box-plot for the log-evidence for TI-BSS-H plotted against the number of chains. As the number of chains increases, the precision in the log-evidence estimate improves. It is also worth noting that the upper end of the range of results does not change very much, but the improvement in precision is seen in the changing of the lower end of the range.

Figure 4.17 shows a box-plot for the run-time plotted against tuples of the number of BSS iterations and the number of total MCMC iterations including leapfrog per temperature. The portion of TI-BSS-H that can be made parallel is the BSS loop. Figure 4.17 illustrates the hit to run time that is incurred if too many interruptions to this loop are made to conduct leapfrog steps.

Figure 4.18 shows a box-plot for the log-evidence for TI-BSS-H plotted against tuples of the number of BSS iterations and the number of total MCMC iterations including leapfrog

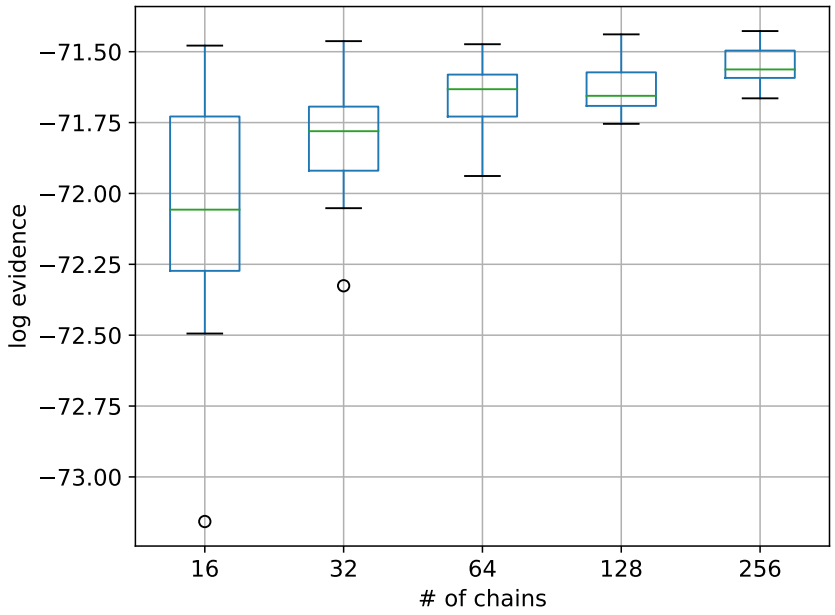


Figure 4.16: Box-plot for log-evidence of TI-BSS-H vs C over 20 runs

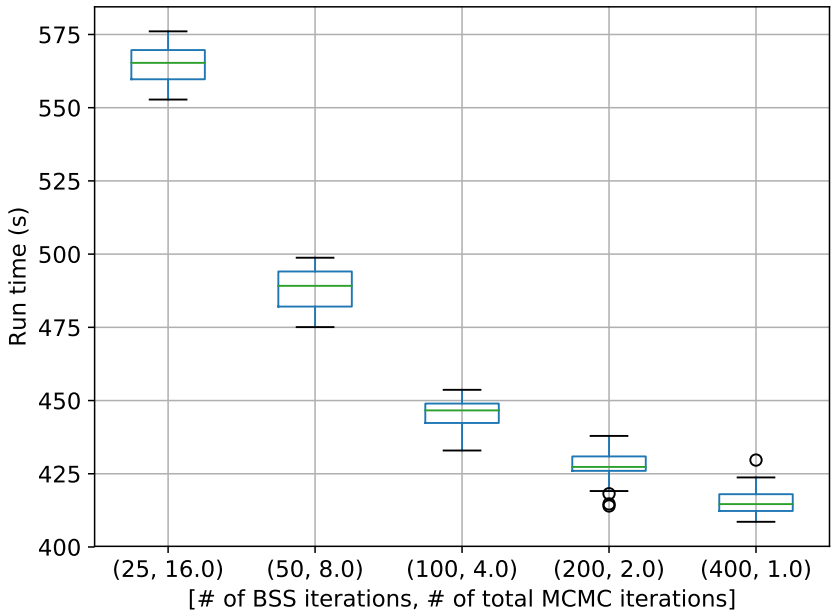


Figure 4.17: Box-plot for run time of TI-BSS-H vs S and M over 20 runs

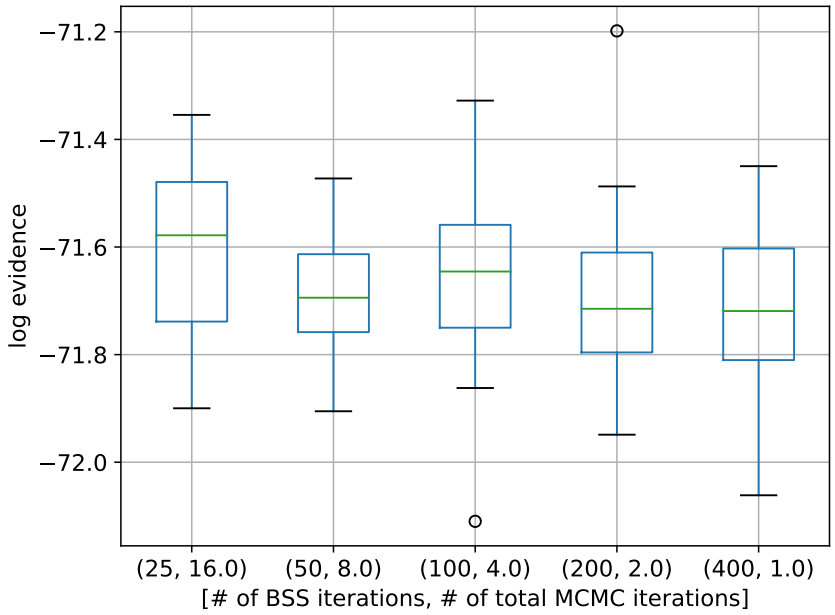


Figure 4.18: Box-plot for log-evidence of TI-BSS-H vs S and M over 20 runs

per temperature. There is no obvious trend here, suggesting that minimal leapfrog steps are necessary to ensure good mixing.

Figure 4.19 shows a box-plot for the run-time plotted against the adaptive annealing control parameter with an exponential fitted curve. This trend and fit in this figure is similar to that observed for TI-Stan in Figure 4.11, and the observations about that figure apply here as well. The fitted curve takes the form

$$T(W) = 0.40 \exp(9.07/W) + 236.54. \tag{4.40}$$

Figure 4.20 shows a box-plot for the log-evidence for TI-BSS-H plotted against the adaptive annealing control parameter. The trend here is again similar to that observed in TI-Stan in Figure 4.12. The precision and mean value of the log-evidence decrease as W is increased.

The analysis in section 4.1 suggest that there should be a quadratic relationship between the run time and the number of bits used in the Hilbert curve transformation

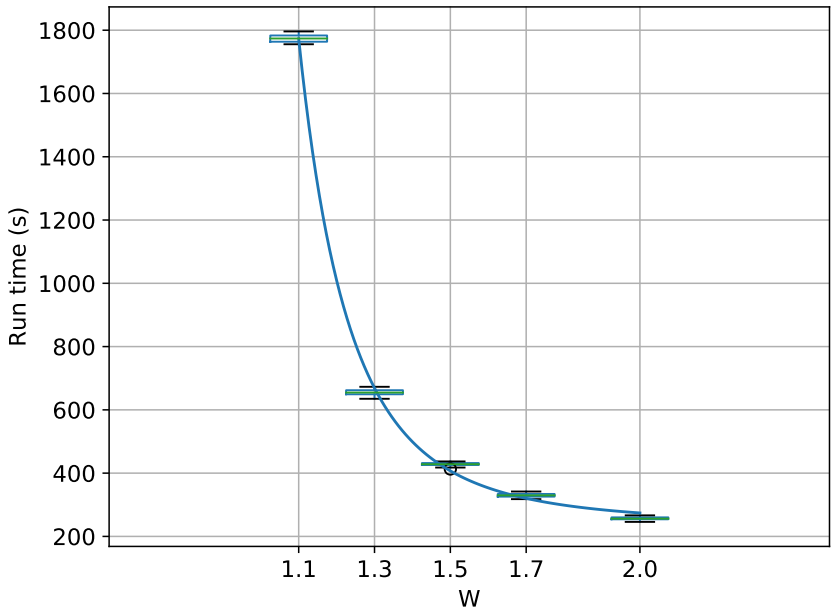


Figure 4.19: Box-plot for run time of TI-BSS-H vs W over 20 runs with an exponential fitted curve

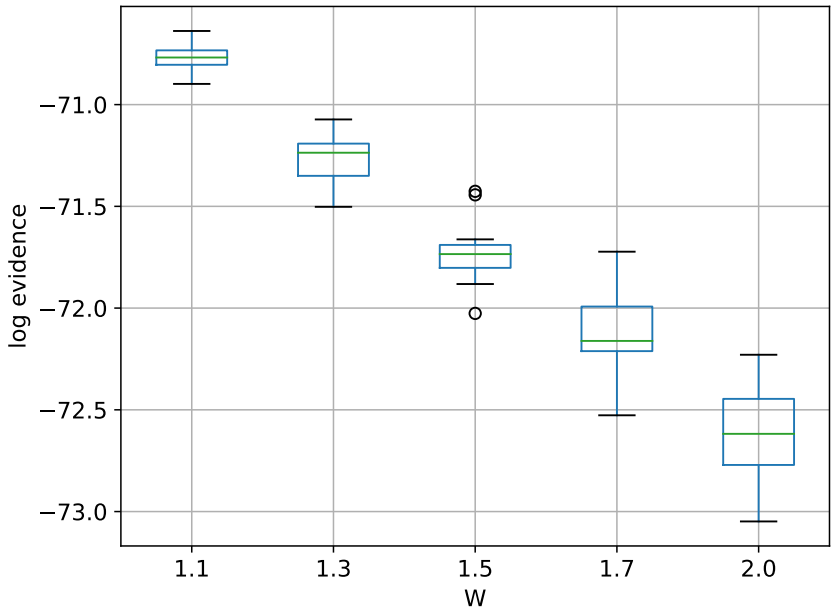


Figure 4.20: Box-plot for log-evidence of TI-BSS-H vs W over 20 runs

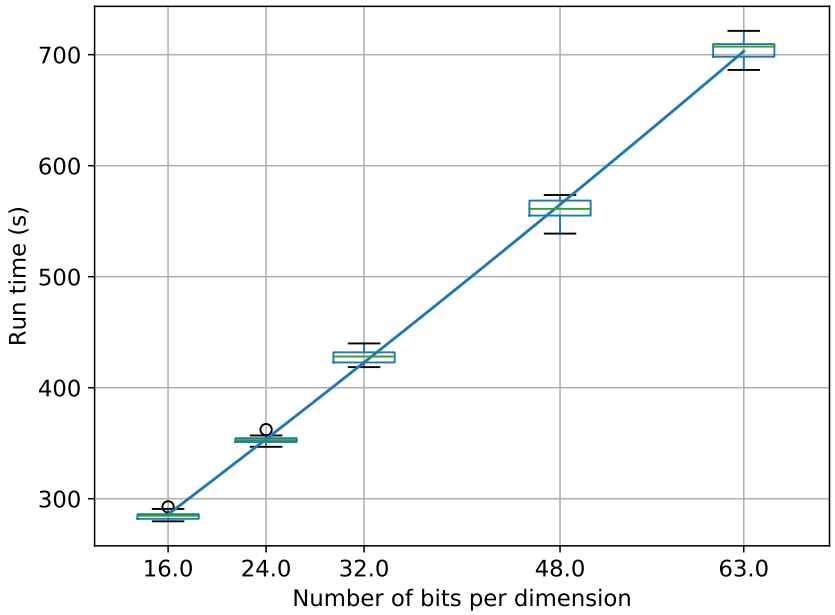


Figure 4.21: Box-plot for run time of TI-BSS-H vs B over 20 runs with a quadratic fitted curve

functions. Figure 4.21 shows a box-plot of the run-time plotted against the number of bits per parameter and a quadratic fitted curve. The fitted curve takes the form

$$T(B) = 0.01B^2 + 8.06B + 154.37. \tag{4.41}$$

The tiny factor on the quadratic term in the curve suggests that at these values, the time complexity is actually mostly linear. Perhaps at much higher value of B, the time complexity would become more obviously quadratic, but values of B greater than 64 are impractical and generally unnecessary.

Figure 4.22 shows a box-plot of the log-evidence for TI-BSS-H plotted against the number of bits per parameter used for the Hilbert curve. The precision in the log-evidence estimate does not meaningfully change for any of these values of B, suggesting that a value of $B = 16$ could actually be used here to no detriment.

Figure 4.23 shows a box-plot of the run-time plotted against the number of available workers and a $\mathcal{O}(1/N)$ fitted curve The fitted curve takes the form,

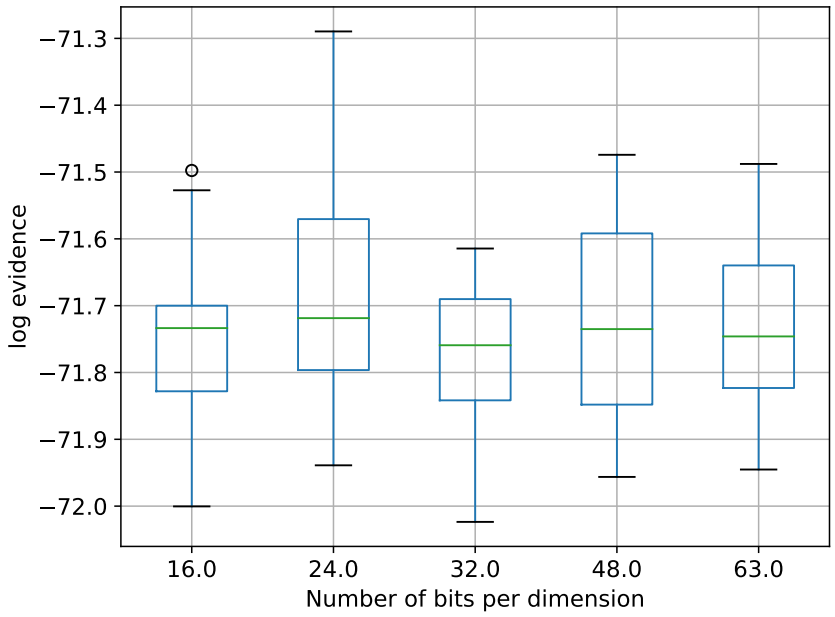


Figure 4.22: Box-plot for log-evidence of TI-BSS-H vs B over 20 runs

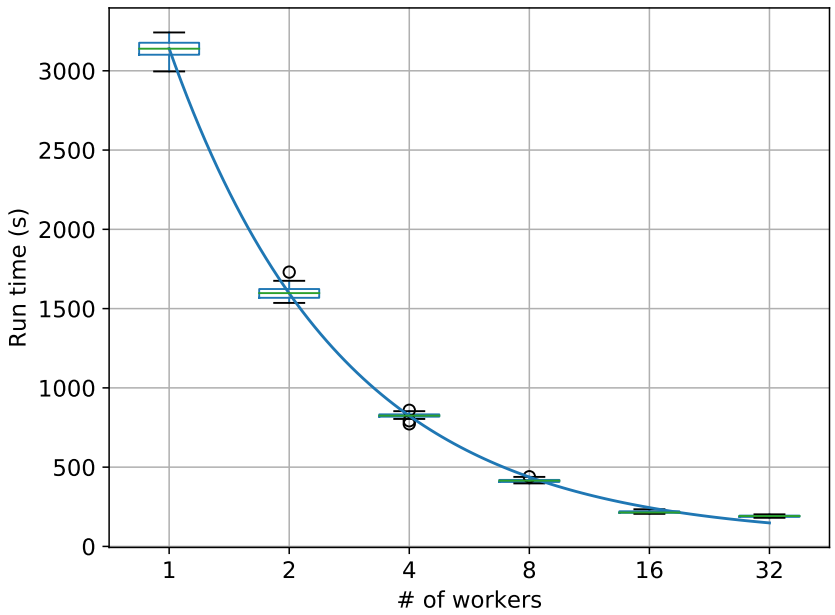


Figure 4.23: Box-plot for run time of TI-BSS-H vs the number of workers over 20 runs with a $1/N$ fitted curve

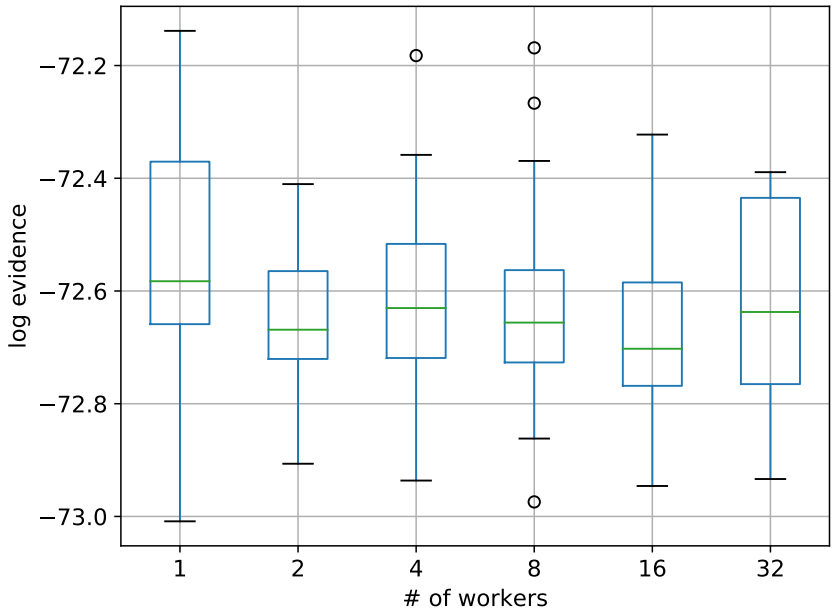


Figure 4.24: Box-plot for log-evidence of TI-BSS-H vs the number of workers over 20 runs

$$T(N) = \frac{3089}{N} + 51.70. \tag{4.42}$$

The trend observed here is similar to that observed for TI-Stan in Figure 4.13: for $C = 64$, a linear speed-up is observed up to 32 workers, confirming the high degree of parallelism that can be achieved with this algorithm.

Figure 4.24 shows a box-plot of the log-evidence for TI-BSS-H plotted against the number of available workers. As expected, there is no systematic change in the log-evidence over this variable.

4.3 Conclusion

This chapter has presented theoretical and empirical time complexity and parallel speed-up analyses of the algorithms described in Chapters 2 and 3. The empirical results have generally agreed with the theoretical results, with the exception of the time complexity of TI-BSS-H as a function of B. These empirical results show how each parameter affects both the precision and run time of each algorithm, and they demonstrate that Combined-

chain nested sampling (NS-CC), Multiple-replacement nested sampling (NS-MR), TI-Stan, and TI-BSS-H can all be parallelized to good effect.

CHAPTER 5

CONCLUSION

Model comparison comprises an important class of inference problems. Computational techniques to solve these problems are time consuming, and several of them can be improved using parallel algorithm design principles. This dissertation has proposed five parallel methods for performing model comparison

- Multiple-replacement nested sampling (NS-MR)
- Combined-chain nested sampling (NS-CC)
- Adaptive thermodynamic integration with binary slice sampling on the Hilbert curve (TI-BSS-H)
- Adaptive thermodynamic integration with binary slice sampling on the Z-order curve (TI-BSS-Z)
- Adaptive thermodynamic integration with Stan (TI-Stan)

It has presented results for each of these methods tested on a set of example problems, ranging from toy problems that illustrate the capabilities of the methods under specific constraints to a signal processing problem using simulated data that demonstrates the methods' capabilities for real-world inference.

It has also analyzed the time complexity of these methods, both theoretically and empirically, over various method parameter values. It has empirically analyzed the parallel speed-up that can be achieved with these techniques, and shown that each one can make effective use of significant parallel computing resources.

The NS-MR technique specifically is useful for situations in which the number of live samples, N, must be high for the sampling of the constrained prior to work well. The NS-CC algorithm is well-suited to use in situations where fewer live samples are required to effectively sample the constrained prior.

TI-BSS-H and TI-Stan work well in general. The TI-based methods would be especially useful in cases where sampling the constrained prior distribution–as nested sampling requires–is difficult. TI-BSS-H does not perform well in problems with very high dimensionality. TI-Stan is preferred for problems with high dimension.

5.1 Future Work

Opportunities for extensions of this work are numerous. These can be split into two broad categories: extensions of these methods onto other parallel hardware architectures and development of additional parallel methods inspired by those presented here.

Regarding extending these methods to new hardware, the most obvious avenues to explore include networked clusters and GPUs. All of the parallel methods presented in this work have been implemented on shared-memory machines (specifically, computers with multi-core processors). What is advantageous or disadvantageous about a particular method would likely change if a different model of parallel computing was used, and those changes are likely worth exploring.

The choices for new methods inspired by those presented here are less obvious. For combined-chain nested sampling, a method that allows the chains to periodically combine their pools of live samples to make the constrained prior sampling more effective. Such a method may not in fact exist, but it could be fruitful to search for it. The multiple stationary frequency detection problem is one that would be well-suited to a trans-dimensional sampling approach rather than a one-model-at-a-time approach like those presented in this work. A trans-dimensional sampling approach (perhaps utilizing jump diffusion sampling) could probably be built using Stan. Development of such a method would be a larger project than

the development of the algorithms presented in this dissertation, as it would likely require significant modifications to Stan itself.

BIBLIOGRAPHY

BIBLIOGRAPHY

Joseph Aczél. *Lectures on functional equations and their applications*, volume 19. Academic press, 1966.

Blaise Barney. Introduction to Parallel Computing, June 2018. URL `https://computing.llnl.gov/tutorials/parallel_comp/`.

Thomas Bayes. A letter from the late Reverend Mr. Thomas Bayes, FRS to John Canton, MA and FRS. *Philosophical Transactions (1683-1775)*, 53:269–271, 1763.

Jakob Bernoulli. *Ars conjectandi*. Impensis Thurnisiorum, fratrum, 1713.

G. Larry Bretthorst. *Bayesian spectrum Analysis and parameter estimation*. Springer-Verlag Berlin Heidelberg, 1988. URL `http://bayes.wustl.edu/glb/book.pdf`.

G. Larry Bretthorst. Nonuniform sampling: Bandwidth and aliasing. *AIP Conference Proceedings*, 567(1):1–28, 2001. doi: 10.1063/1.1381847. URL `http://scitation.aip.org/content/aip/proceeding/aipcp/10.1063/1.1381847`.

Brendon J. Brewer, Livia B. Pártay, and Gábor Csányi. Diffusive Nested Sampling. *Statistics and Computing*, 21(4):649–656, 2011. doi: 10.1007/s11222-010-9198-8.

Nikolas S. Burkoff, Csilla Várnai, Stephen A. Wells, and David L. Wild. Exploring the Energy Landscapes of Protein Folding Simulations with Bayesian Computation. *Biophysical Journal*, 102(4):878–886, February 2012. ISSN 0006-3495. doi: 10.1016/j.bpj.2011.12.053. URL `http://dx.doi.org/10.1016/j.bpj.2011.12.053`.

B. L. Burrows. A New Approach to Numerical Integration. *Journal of the Institute of Mathematics and its Applications*, 26:151–173, 1980.

Ben Calderhead and Mark Girolami. Estimating Bayes factors via thermodynamic integration and population MCMC. *Computational Statistics & Data Analysis*, 53(12): 4028–4045, October 2009. ISSN 01679473. doi: 10.1016/j.csda.2009.07.025. URL `https://linkinghub.elsevier.com/retrieve/pii/S0167947309002722`.

Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan : A Probabilistic Programming Language. *Journal of Statistical Software*, 76(1), 2017. ISSN 1548-7660. doi: 10.18637/jss.v076.i01. URL `http://www.jstatsoft.org/v76/i01/`.

Rajiv Ch and rasekaran. INTEL TO UNVEIL FASTER PENTIUM CHIP. *Washington Post*, January 1997. ISSN 0190-8286. URL https://www.washingtonpost.com/archive/business/1997/01/08/intel-to-unveil-faster-pentium-chip/9d6bdbd2-51b0-4a56-b24a-f4b1ee8b795e/.

Richard Threlkeld Cox. Probability, frequency, and reasonable expectation. *American Journal of Physics*, 17:1–13, 1946.

F. Feroz, M. P. Hobson, and M. Bridges. MultiNest: an efficient and robust Bayesian inference tool for cosmology and particle physics. *Monthly Notices of the Royal Astronomical Society*, 398(4):1601–1614, October 2009. ISSN 1365-2966. doi: 10.1111/j.1365-2966.2009.14548.x. URL http://dx.doi.org/10.1111/j.1365-2966.2009.14548.x.

N. Friel and A. N. Pettitt. Marginal likelihood estimation via power posteriors. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(3):589–607, July 2008. ISSN 1369-7412, 1467-9868. doi: 10.1111/j.1467-9868.2007.00650.x. URL http://doi.wiley.com/10.1111/j.1467-9868.2007.00650.x.

Gabriel. How to compute a 3d Morton number (interleave the bits of 3 ints), August 2013. URL https://stackoverflow.com/revisions/18528775/2. Published: Stackoverflow.

Andrew Gelman and Xiao-Li Meng. Simulating normalizing constants: From importance sampling to bridge sampling to path sampling. *Statistical science*, 13(2):163–185, 1998.

Paul M Goggans and Ying Chi. Using thermodynamic integration to calculate the posterior probability in Bayesian model selection problems. *AIP Conference Proceedings*, 707(1): 59–66, 2004. doi: 10.1063/1.1751356.

Peter J. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732, 1995. doi: 10.1093/biomet/82.4.711.

Phil Gregory. *Bayesian Logical Data Analysis for the Physical Sciences*. Cambridge University Press, paperback edition, 2010.

W.J. Handley, M.P. Hobson, and A.N. Lasenby. PolyChord: next-generation nested sampling. *Monthly Notices of the Royal Astronomical Society*, 453(4):4384–4398, November 2015. doi: 10.1093/mnras/stv1911.

Wilfred Keith Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

R. Wesley Henderson and Paul M. Goggans. Parallelized Nested Sampling. *AIP Conference Proceedings*, 1636(1):100–105, 2014. doi: 10.1063/1.4903717.

R. Wesley Henderson and Paul M. Goggans. Using the Z-order curve for Bayesian model comparison. In A. Polpo, J. Stern, F. Louzada, R. Izbicki, and H. Takada, editors, *Bayesian Inference and Maximum Entropy Methods in Science and Engineering. MaxEnt 2017. Springer Proceedings in Mathematics & Statistics*, volume 239, pages 295–304, 2018.

R. Wesley Henderson, Paul M. Goggans, and Lei Cao. Combined-chain nested sampling for efficient Bayesian model comparison. *Digital Signal Processing*, 70:84–93, November 2017. doi: 10.1016/j.dsp.2017.07.021. URL http://www.sciencedirect.com/science/article/pii/S1051200417301719.

Matthew D. Hoffman and Andrew Gelman. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15: 1593–1623, April 2014.

Edwin T. Jaynes. How Does the Brain Do Plausible Reasoning? Technical report, Stanford Microwave Laboratory, 1957.

Edwin T. Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003.

S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, May 1983.

John G. Kirkwood. Statistical Mechanics of Fluid Mixtures. *The Journal of Chemical Physics*, 3(5):300–313, 1935.

Pierre-Simon Laplace. Mémoire sur la probabilité des causes par les évènemens. *Mémoires de l'Académie royale des sciences*, 6:621–656, 1774.

Jun S. Liu, Rong Chen, and Tanya Logvinenko. A Theoretical Framework for Sequential Importance Sampling with Resampling. In Arnaud Doucet, Nando de Freitas, and Neil Gordon, editors, *Sequential Monte Carlo Methods in Practice*, Statistics for Engineering and Information Science, pages 225–246. Springer New York, New York, NY, 2001. ISBN 978-1-4757-3437-9. doi: 10.1007/978-1-4757-3437-9_11. URL https://doi.org/10.1007/978-1-4757-3437-9_11.

Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

Radford M. Neal. Probabilistic Inference Using Markov Chain Monte Carlo Methods. Technical Report CRG-TR-93-1, University of Toronto, September 1993. URL https://bayes.wustl.edu/Manual/RadfordNeal.review.pdf.

Radford M. Neal. Slice Sampling. *The Annals of Statistics*, 31(3):705–767, 2003. doi: 10.1214/aos/1056562461. URL http://projecteuclid.org/euclid.aos/1056562461.

Radford M. Neal. MCMC using Hamiltonian dynamics. In Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng, editors, *Handbook of Markov Chain Monte Carlo*. Chapman & Hall / CRC Press, 2011. URL http://arxiv.org/abs/1206.1901. arXiv: 1206.1901.

Chris J. Oates, Theodore Papamarkou, and Mark Girolami. The Controlled Thermodynamic Integral for Bayesian Model Evidence Evaluation. *Journal of the American Statistical Association*, 111(514):634–645, April 2016. ISSN 0162-1459, 1537-274X. doi:

10.1080/01621459.2015.1021006. URL https://www.tandfonline.com/doi/full/10.1080/01621459.2015.1021006.

Yosihiko Ogata. A Monte Carlo method for high dimensional integration. *Numerische Mathematik*, 55(2):137–157, March 1989. ISSN 0945-3245. doi: 10.1007/BF01406511. URL https://doi.org/10.1007/BF01406511.

David B. Phillips and Adrian F. M. Smith. Bayesian model comparison via jump diffusions. In W.R. Gilks, S. Richardson, and D.J. Spiegelhalter, editors, *Markov chain Monte Carlo in practice*. Chapman & Hall, 1996.

George Pólya. *Mathematics and Plausible Reasoning*, volume 1 and 2. Princeton University Press, 1954.

Sheldon M. Ross. *Introduction to Probability Models*. Academic Press, tenth edition, 2010.

Hans Sagan. *Space-Filling Curves*. Springer-Verlag New York, 1994.

S. V. Semenovskaya, K. A. Khachaturyan, and A. G. Khachaturyan. Statistical mechanics approach to the structure determination of a crystal. *Acta Crystallographica Section A: Foundations of Crystallography*, 41(3):268–273, 1985. doi: 10.1107/S0108767385000563.

Devinder S. Sivia and John Skilling. *Data Analysis: a Bayesian Tutorial*. Oxford University Press, second edition, 2006.

John Skilling. *BayeSys and MassInf*. Maximum Entropy Data Consultants Ltd., February 2004a.

John Skilling. Nested sampling. *Bayesian inference and maximum entropy methods in science and engineering*, 735(1):395–405, 2004b. doi: 10.1063/1.1835238. URL http://aip.scitation.org/doi/abs/10.1063/1.1835238.

John Skilling. Programming the Hilbert curve. In G. Erickson and Y. Zhai, editors, *Bayesian Inference and Maximum Entropy Methods in Science and Engineering: 23rd International Workshop*, pages 381–387. American Institute of Physics, 2004c.

John Skilling. Nested sampling for general Bayesian computation. *Bayesian Analysis*, 1(4):833–859, 2006.

John Skilling. Bayesian Computation in Big Spaces – Nested Sampling and Galilean Monte Carlo. *AIP Conference Proceedings*, 1443(1):145–156, 2012.

John Skilling and Davd J. C. MacKay. [Slice Sampling]: Discussion. *The Annals of Statistics*, 31(3):753–755, 2003. URL http://www.jstor.org/stable/3448417.

Stan Development Team. PyStan: the Python interface to Stan, 2018. URL http://mc-stan.org.

Howard M. Taylor and Samuel Karlin. *An Introduction to Stochastic Modeling*. Academic Press, third edition, 1998.

Henk C. Tijms. *A First Course in Stochastic Models*. John Wiley & Sons, 2003.

Eric W. Weisstein. Erlang Distribution. From MathWorld–A Wolfram Web Resource., February 2016. URL `http://mathworld.wolfram.com/ErlangDistribution.html`.

Albert Yu. The Story of Intel MMX$^{\text{TM}}$ Technology. *Intel Technology Journal*, Q3:1–2, 1997.

V. Černý. Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, January 1985.

VITA

R. Wesley Henderson was born and raised in Ruston, Louisiana. He graduated *magna cum laude* from Louisiana Tech University in 2011 with a bachelor of science degree in civil engineering. He graduated from Rensselaer Polytechnic Institute in 2012 with a master of science degree in architectural sciences with a concentration in acoustics. There, under the guidance of Dr. Ning Xiang, he investigated the application of Bayesian inference to room acoustic modal analysis.

He is currently a doctoral candidate in the electrical engineering program at the University of Mississippi, working with adviser Prof. Paul M. Goggans. His research interests include the development and application of numerical inference techniques, specifically using nested sampling, to various engineering domains, including acoustics and signal processing.